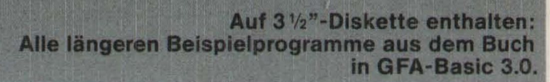
Markt&Technik

Jens Muus
Wilhelm Besenthal

ATARI ST
PROGRAMMIERPRAXIS GFA-BASIC 3.0

Das Handbuch zur professionellen Programmentwicklung mit GFA-Basic:
★ Grundlagen ★ Menüerstellung
★ GEM-Programmierung ★ Betriebssystemprogrammierung ★ 3-D-Grafik
★ Raytracing und vieles mehr





Atari ST Programmierpraxis
GFA-BASIC 3.0

Jens Muus · Wilhelm Besenthal

ATARI ST PROGRAMMIERPRAXIS

GFA-BASIC 3.0

Das Handbuch zur professionellen Programmentwicklung mit GFA-Basic:
★ Grundlagen ★ Menüerstellung ★ GEM-Programmierung
★ Betriebssystemprogrammierung ★ 3-D-Grafik
★ Raytracing ★ und vieles mehr

Markt&Technik Verlag AG

CIP-Titelaufnahme der Deutschen Bibliothek

Muus, Jens:
ATARI-ST-Programmierpraxis GFA-BASIC 3.0 :
d. Handbuch zur professionellen Programmentwicklung mit GFA-Basic:
Grundlagen, Menüerstellung, GEM-Programmierung,
Betriebssystemprogrammierung, 3-D-Grafik, Raytracing und vieles mehr /
Jens Muus ; Wilhelm Besenthal. –
Haar bei München : Markt-u.-Technik-Verl., 1989
ISBN 3-89090-702-4
NE: Besenthal, Wilhelm

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

92 91 90 89

ISBN 3-89090-702-4

Einbandgestaltung: Grafikdesign Heinz Rauner
Dieses Buch wurde mit Desktop-Publishing-Programmen bearbeitet
und über Linotronic 300 belichtet.
Druck: Pustet, Regensburg
Printed in Germany

Inhaltsverzeichnis

	Vorwort	7
1	**Einführung**	9
1.1	Der Editor	9
1.1.1	Die Menüzeile	10
1.2	Prozeduren und Funktionen	13
1.2.1	Übergabe von Parametern	14
1.2.2	VAR-Parameter	14
1.3	Programmschleifen	15
1.4	Entscheidungsanweisungen	16
2	**Wissenswertes zur Programmierarbeit**	21
2.1	Die Speicherverwaltung	21
2.2	Arbeiten mit mehreren Bildschirmen	23
2.3	Geräusche und Musik ohne Programmunterbrechung	26
3	**Die GEM-Programmierung**	35
3.1	Die Drop-down-Menüs	36
3.2	Ereignisse unter GEM	40
3.3	Dialogboxen	53
3.3.1	Objektstrukturen	54
4	**Die Fensterverwaltung**	73
4.1	Grundlagen	75
4.2	Der Message-Buffer	80
4.3	Fensterprogrammierung im Programm	82
4.3.1	Neuzeichnen von Bildschirmteilen	92
4.3.2	Die anderen Nachrichten	96
5	**Diskettenbefehle**	107
5.1	Dateiverwaltung	112
5.2	Grundlagen	112
5.3	Allgemeine Dateibefehle	114
5.4	Befehle und Funktionen für sequentielle Dateien	117
5.5	Befehle für relative Dateien	120
5.6	Dateibefehle im Einsatz	122

6	**Grafik**	133
6.1	Der Bildschirmausgabeteil des Atari ST	134
6.2	Farbgrafik	137
6.2.1	GRAPHMODE bei der Farbgrafik	139
6.2.2	Beispielprogramme	141
6.3	Die Grafikbefehle	145
6.3.1	Die Befehle GET und PUT	150
6.4	Räumliche Darstellungen	154
6.4.1	Raytracing	156
6.4.2	Fraktale	176
6.4.3	Aussichten	199
6.5	Objektorientiertes Zeichenprogramm	199
7	**Fortgeschrittene Programmierung**	217
7.1	Erweiterte Grafikfunktionen durch GDOS	217
7.1.1	Neue Zeichensätze	218
7.2	Der Befehl RC_COPY	238
7.3	Aufruf von Maschinenprogrammen	240
8	**Das Resource Construction Set**	251
	Anhang	257
A	Die Befehle des GFA-Basic 3.0	257
B	Füll-, Linienmuster, Textstil und Maussymbole	317
C	Steuersequenzen des VT52-Emulators	323
D	Anschlußbelegungen des Atari ST	327
	Stichwortverzeichnis	333
	Hinweise auf weitere M&T-Produkte	338



Vorwort

Der Atari ST ist seit dem Jahre 1985 in der Bundesrepublik erhältlich. Ausgestattet mit einer Hardware, die auch heute noch vielen Konkurrenten das Fürchten lehrt, haben ihm seine Väter leider nur eine sehr unvollkommene Programmiersprache, das damalige ST-Basic, mit auf den Weg gegeben. Zwar hat das ST-Basic im Laufe seines Daseins noch einige Überarbeitungen erfahren, fehlerfrei und schnell ist es trotzdem nicht geworden. Mitte 1988 wurde es schließlich durch das wesentlich bessere Omikron-Basic ersetzt.

Ganz anders verlief die Geschichte des GFA-Basic. Die Version 1.0 war in der ersten Hälfte des Jahres 1986 erhältlich und lieferte von Anfang an, trotz kleinerer Fehler, sehr gute Ergebnisse. Die Fehler wurden ausgemerzt, Ende 1986 erschien die erweiterte Version 2.0. Durch einen angemessenen Preis und dem kundenfreundlichen Upgrade-Service der Firma, wurde diese Version bald zur Standard-Programmiersprache des Atari ST. Der ebenfalls erhältliche Compiler, der das ohnehin nicht langsame Basic um ein Vielfaches beschleunigte, rundete das positive Bild ab.

Seit Mitte 1988 ist GFA-Basic 3.0 erhältlich. Es ist mit einer so großen Vielfalt an Befehlen ausgestattet, daß vielen ST-Besitzern »Angst und Bange« (gemacht) wurde. In einigen Fachzeitschriften wurde (wird) GFA-Basic 3.0 als »überladen« dargestellt – ein Urteil, dem sich die Autoren nach vielen Stunden Programmierarbeit nicht anschließen können.

Der schon vorher sehr gute Editor wurde nochmals überarbeitet und erhielt dabei einige sinnvolle Erweiterungen. Alle AES-Funktionen sind über Basic-Befehle erreichbar, d.h., Dialogboxen und Fenster lassen sich ab sofort recht einfach programmieren bzw. verwalten. Programme der Version 2.0 sind, bis auf Kleinigkeiten, voll kompatibel zur Version 3.0.

Das vorliegende Buch soll keine Einführung in das GFA-Basic 3.0 darstellen. Vielmehr soll es als Arbeitsgrundlage für fortgeschrittene Programmierung dienen. Es werden viele Themen beschrieben, an die sich nicht jeder Programmierer sofort heranwagt. Der aufmerksame Leser wird in den einzelnen Kapiteln immer wieder Routinen entdecken, die er sofort, oder mit kleinen Änderungen versehen, in seine Programme einbauen kann.

Zu den Programmen

Sie finden alle längeren Programme auf der beiliegenden Diskette. Das spart Ihnen nicht nur Tipparbeit, sondern ermöglicht zugleich, daß Sie sich auf das Wesentliche eines Themas konzentrieren können. Die Programmnamen enthalten immer die entsprechende Kapitelnummer und sind innerhalb eines Kapitels durchnumeriert. Mehrere Programme eines Kapitels sind in einem Ordner zusammengefaßt. Die Programme sind nicht als Komplettlösungen zu verstehen, sie sollen vielmehr Zusammenhänge erkennbar machen und schnell eigenen Bedürfnissen anpaßbar sein. In einigen Fällen wurde auch bewußt nicht »optimal« programmiert. Die Geschwindigkeit der Programme hat bei der Programmentwicklung oftmals eine untergeordnete Rolle gespielt, wichtiger war die Nachvollziehbarkeit der einzelnen Programmsequenzen.

Einige Programme, die Daten nachladen, enthalten einen bereits voreingestellten Suchpfad. Dieser muß von Ihnen gegebenenfalls, nach dem Kopieren der Programme und Dateien in andere Ordner, angepaßt werden.

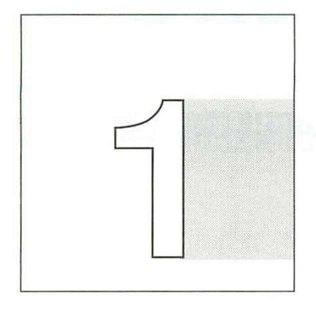
Alle Programme wurden auf verschiedenen ST-Versionen mit monochromem Monitor überprüft und sollten daher lauffähig sein. Beachten Sie bitte auch die Hinweise auf Speicherumfang und Gerätekonfiguration in den Kapiteln.

Vielleicht interessiert Sie, lieber Leser, mit welcher Geräteausstattung dieses Buch entstand. Erstellt wurde das Buch auf einem Atari 520 ST+ und einem Mega ST2. Beide Autoren verfügen über die Atari Festplatte SH204. Ein Autor nennt einen OKI Laserdrucker und einen 9-Nadel-Drucker (OKI ML192 Elite) sein eigen, der andere einen 24-Nadel-Drucker (OKI ML390). Die Matrixdrucker wurden für schnelle Listing- und Korrekturausdrucke eingesetzt, der Laserdrucker erstellte den Ausdruck für den Verlag und die Hardcopys. Für die Textverarbeitung wurde das Programm »1st Word Plus« verwendet. Hilfreiche Dienste verrichtete auch das Programm »Twist« aus dem Markt&Technik-Verlag. Es ermöglicht, durch Umschalten, das Arbeiten mit verschiedenen Programmen, die zur gleichen Zeit im Speicher stehen.

Bedanken möchten wir uns an dieser Stelle bei Herrn Grolms vom Markt&Technik Verlag und Frau Spacey-Rennings, die uns bei der Erstellung des Ihnen vorliegenden Endprodukts tatkräftig unterstützt haben. Besonderer Dank gilt auch unseren »Mädels«, die es während des Schreibens nicht leicht mit uns hatten.

Wir wünschen Ihnen viel Spaß beim Lesen und viel Erfolg bei der Verwirklichung eigener Programmideen.

Wilhelm Besenthal / Jens Muus



Einführung

Mit der ersten Version des GFA-Basic kam 1986 ein völlig neu entwickelter Basic-Interpreter auf den Markt, der eine neue Ära in der Basic-Programmierung auf dem Atari ST einleitete. Der auffallendste Unterschied zu anderen Basic-Dialekten (zum Beispiel dem »berühmt-berüchtigten« ST Basic) war das Fehlen der sonst üblichen Zeilennummern. Endlich wurde von diesen überflüssigen Bestandteilen der Programme Abstand genommen. Der Programmierer war nun auch in Basic, wie schon vorher in anderen Hochsprachen, auf einen bestimmten Programmierstil angewiesen: *die strukturierte Programmierung*. In geradezu hervorragender Weise unterstützt GFA-Basic diesen Programmierstil – es zwingt den Programmierer quasi dazu, seine Programme übersichtlich zu gestalten. Wer sich einmal daran gewöhnt hat, wird nicht mehr anders programmieren wollen. Der sogenannte »Spaghetti-Code«, der in vielen Programmen vorherrschte und seinem Namen alle Ehre machte, ist mit einer modernen Hochsprache wie dem GFA-Basic zwar noch möglich, aber geradezu »schwierig« zu programmieren. Es ist nur ein Befehl je Programmzeile erlaubt. Programmschleifen werden im Listing automatisch eingerückt, was schon eine rein optische Übersichtlichkeit eines Programms bringt.

Diese und andere Besonderheiten des GFA-Basic 3.0 möchten wir in diesem Kapitel in loser Reihenfolge beschreiben. Zunächst jedoch einige Worte zum Editor und seinen Besonderheiten.

1.1 Der Editor

Ein herausragendes Merkmal des GFA-Basic 3.0 ist der komfortable Editor. Hier wurde auf die Bedürfnisse des Programmierers besonderes Augenmerk gerichtet. Es macht wirklich Spaß, mit diesem Editor Programme zu entwickeln, da keine unnötigen oder umständlich zu bedienenden Funktionen die Arbeit behindern. Alle Funktionen können sowohl mit der Maus als auch über die Tastatur aktiviert werden. Über eventuelle Tippfehler (der Befehle) werden Sie sofort informiert. Eine Programmzeile mit fehlerhaft eingegebenen Befehlen kann darüber hinaus überhaupt nicht in das Programm übernommen werden, da der Cursor so lange in dieser Zeile stehenbleibt, bis die penibel arbeitende Syntaxkontrolle des Editors keinen Fehler mehr feststellt.

```
 | Save |Save,A| Quit | New  |BlkSta|Replac| Pg ⇧ |Txt 16|Direct| Run  |
 | Load |Merge |Llist |Block |BlkEnd| Find | Pg ⇩ |Insert| Flip | Test|
RESERVE FRE(i)-256000
IF GDOS?
  a%=VST_LOAD_FONTS(0)
  FOR i%=0 TO a%+1
    f%=VQT_NAME(i%,f$)
    PRINT AT(1,i%*4);f$''"Kennung: ";f%''"Fontnummer: ";i%
    DEFTEXT ,0,,15,f%
    TEXT 80,i%*4*16+20,"Probetext - 1234567890"
  NEXT i%
  ~VST_UNLOAD_FONTS(0)
ELSE
  ALERT 2,"GDOS fehlt!",1,"ENDE",var
ENDIF
RESERVE FRE(i)+256000
```

Bild 1.1: Der Editor von GFA 3.0

1.1.1 Die Menüzeile

Am oberen Bildschirmrand ist die sogenannte Menüzeile. Sie besteht genaugenommen aus zwei Zeilen mit diversen Funktionen, die durch Anklicken mit der Maus aktiviert werden. Alternativ dazu können alle Funktionen auch mit den Funktionstasten der Tastatur aufgerufen werden. Die Einteilung auf dem Bildschirm bietet eine Hilfe dazu an, mit welcher Taste welche Funktion erreicht wird. Das beginnt links mit »**Load**« und »**Save**«, aufzurufen mit [F1] und [Shift]+[F1] und endet rechts mit »**Test**« und »**Run**«, aufzurufen mit [F10] und [Shift]+[F10]. So wird zum Beispiel die Funktion »**Find**« mit [F6] aufgerufen.

Durch die Felder »**Load**« und »**Save**« werden die GFA-Basic-Programme geladen bzw. gespeichert. Dabei muß folgendes beachtet werden: Ab GFA-Basic Version 3.04 sind die »**Load**«- und »**Save**«-Routinen geändert worden. Die gespeicherten Programme benötigen dadurch etwa 15% weniger Platz auf den Disketten. Neben diesem Vorteil hat das allerdings auch einen nicht zu übersehenden Nachteil. Die so gespeicherten Programme können nämlich mit den GFA-Basic-Versionen vor 3.03 nicht eingeladen werden. Sie werden einfach mit einer Fehlermeldung abgewiesen. Das ist natürlich nicht sonderlich schlimm, denn wer eine neuere Version des GFA-Basic sein eigen nennt, wird später sicher nicht mit älteren Versionen arbeiten wollen. Ganz anders wird die Sache, wenn der Programmierer sein Programm weitergibt oder verkauft. Es sollte dann natürlich sichergestellt sein, daß der Empfänger dieses Programms auch die neuere Programmversion des GFA-Basic besitzt. Andernfalls ist das Programm zunächst für ihn unbrauchbar.

Es gibt allerdings auch hier wieder eine Möglichkeit, dieses Problem über einen Umweg zu beseitigen. Das Programmfile muß, um geladen werden zu können, durch Null-Bytes verlängert werden.

```
OPEN "A",#1,dateiname$
PRINT #1,STRING$(1000,0)
CLOSE #1
```

Es kann hierbei allerdings nicht genau gesagt werden, um wie viele Bytes das Programm länger werden muß. Eventuell müssen Sie die oben angeführte Routine mehrmals aufrufen, bis das Programm endlich eingeladen werden kann.

Eine andere Möglichkeit besteht darin, das Programm mit »**Save, A**« als ASCII-File abzuspeichern. Dieses File kann dann auch mit den älteren Versionen (sogar mit den Versionen 1.xx und 2.xx, sofern keine speziellen 3.0-Befehle enthalten sind) mit »**Merge**« geladen werden. Einfacher ist es natürlich, die neueste GFA-Basic-Version zu besorgen. Erkundigen Sie sich bei der Firma GFA in Düsseldorf, wie Sie Ihre ältere Version umtauschen können. Wir haben dabei gute Erfahrungen gemacht. Unsere Version 3.0 wurde stets kostenlos (von den Portokosten abgesehen) und sehr schnell umgetauscht.

Die Funktionen »**Merge**« und »**Save, A**« haben wir schon angesprochen. »**Save, A**« speichert einen Programmtext als ASCII-File auf Diskette. Dabei werden im Programm eingeklappte Prozeduren aufgeklappt. Die Kennzeichnung derartiger Prozeduren durch das Größer-als-Zeichen »>« bleibt dabei jedoch erstaunlicherweise erhalten. Es dient dazu, zuvor eingeklappte Prozeduren nach dem »**Merge**« wieder eingeklappt darzustellen.

Das Feld »**Llist**« ermöglicht die Ausgabe des Programmlistings auf einen angeschlossenen Drucker. Es können hierbei auch diverse Steuerbefehle an den Drucker übergeben werden. Dazu werden die Befehle einfach in den zu druckenden Programmtext geschrieben. Der Befehl »*.PL xx*« dient übrigens nur dazu, die maximale Seitenlänge zu bestimmen. Die weiteren Funktionen dieses Befehls, die in der Anleitung zum GFA-Basic aufgeführt sind, sind nicht möglich. Zusätzlich zu den in der GFA-Basic-Anleitung genannten Befehlen gibt es noch:

.PA

Führt einen Seitenvorschub durch.

.P–
.P+

Die Punktbefehle werden nicht gelistet (–) bzw. gelistet (+). Dabei gelten ».P+ « und ».P– « für das gesamte Listing (wie ».Nx«).

Das Feld »**Block**« dient dazu, die Blockfunktionen für einen zuvor definierten Programmausschnitt (mit »**BlkSta**« und »**BlkEnd**« an der gewünschten Cursorposition) aufzurufen. Sie erhalten dann ein weiteres Auswahlmenü in der ersten Bildschirmzeile. Beachten Sie dabei, daß sich Positionen immer auf die aktuelle Position des Cursors beziehen. Grundsätzlich können nur komplette Programmzeilen als Block markiert werden. Die Markierung ist in Bild 1.1 gut zu sehen. Der Block wird durch das Raster hervorgehoben.

Eine etwas verunglückte Reaktion liefern »**Txt 16**« und »**Txt 8**«, weil die Cursorposition beim Umschalten zwischen diesen Darstellungsgrößen des Listings leider nicht beibehalten wird. Wenn auch die kleinere Textdarstellung des Listings durch die geringere Auflösung etwas schlechter zu lesen ist, erhält man dadurch jedoch einen größeren Überblick über das Listing. Vielleicht wird das Problem mit dem Cursor ja auch noch gelöst.

»**Replace**« und »**Find**« dienen dazu, bestimmte Zeichen oder Wörter zu suchen und eventuell durch andere zu ersetzen. Das funktionierte bei unseren Tests leider nicht über eingeklappte Prozeduren.

Über »**Flip**« können Sie in den Ausgabebildschirm GFA 3.0 gelangen. Dabei wird der Mauscursor abgeschaltet. So kann man den letzten Bildschirminhalt des Programms noch einmal ansehen. Wird dagegen »**Direct**« gewählt, gelangen Sie ebenfalls in den Ausgabebildschirm, hier wird dann allerdings der *Direktmodus* eingeschaltet. Jetzt können Sie Befehle und Funktionen direkt, also ohne den Umweg über ein Basic-Programm angeben. Nach dem Druck der [Return]-Taste werden die Angaben auf Syntax überprüft und dann sofort ausgeführt. Durch den Befehl »EDIT« kommen Sie wieder in den Editor.

Ganz links oben in der GFA-Basic-Menüzeile sehen Sie das Atari-Zeichen. Wenn dieses angeklickt wird, erscheint eine GEM-übliche Menüzeile. So kann man auch während der Programmierung die eventuell vorhandenen Accessories erreichen.



Bild 1.2: GFA-Menüzeile

In der schwarzen Fläche unter dem Atari-Zeichen zeigen zwei Zeichen (nach oben gerichteter Pfeil bzw. das Exponent-Zeichen) den Zustand der [CapsLock]-Taste und der [NumLock]-Tastenkombination [Control]+[(] oder [Control]+[-] (»Klammer auf«- und »Minus«-Zeichen des Zifferntastenblocks) an. Sie können durch das Anklicken der Fläche unter dem Atari-Zeichen diese beiden Funktionen ebenfalls ein- und ausschalten. [NumLock] ersetzt das Drücken der [Control]-Taste, um die Sonderfunktionen des Ziffernzahlenblocks zu erreichen.

Oben rechts wird die Uhrzeit eingeblendet. Dabei handelt es sich um die Systemzeit Ihres Rechners, die Sie hier auch einstellen können, wenn Sie das Feld mit der Maus anklicken. Darunter ist eine Zahl sichtbar, die angibt, in welcher Programmzeile sich der Cursor befindet. Wenn auch das GFA-Basic ohne Zeilennummern auskommt, ist es doch manchmal nötig, die Position des Cursors innerhalb eines Programmtextes zu erfahren. Aus eigener Erfahrung können wir hierzu allerdings anmerken, daß auch diese Angabe der Zeilennummer kaum beachtet wird, wenn man von den Möglichkeiten Gebrauch macht, die das Basic liefert. Wer seine Routinen zum Beispiel in Prozeduren faßt und diese »eingeklappt« (siehe Abschnitt 1.2) darstellt, erhält automatisch eine größere Übersicht über sein Programm.

1.2 Prozeduren und Funktionen

Bei den Prozeduren handelt es sich um Unterprogramme. Diese sind mit einem Namen, einem *Label*, versehen, der auch zum Aufruf der Unterprogramme benötigt wird.

```
PROCEDURE name
  ...
  Programmtext
  ...
RETURN
```

Die Prozeduren müssen immer im Hauptprogramm stehen. Es kann also keine Prozedur innerhalb einer anderen definiert werden. Zum Aufruf einer Prozedur stehen mehrere Möglichkeiten offen:

GOSUB name oder
@name oder
name

Es reicht also schon die einfache Angabe des Prozedurnamens, um diese aufzurufen.

Innerhalb einer Prozedur können Sie genau wie im Hauptprogramm alle Variablen nutzen. Es gibt allerdings auch die Möglichkeit, sogenannte *lokale Variablen* zu definieren. Geben Sie dazu einfach am Beginn der Prozedur den Befehl

LOCAL variable[,variable2,variable3]

an. Es können auch mehrere Variablennamen, durch Kommas getrennt, angegeben werden. Von nun an sind die angegebenen Variablen nur in dieser Prozedur sowie in den von dieser Prozedur aufgerufenen anderen Prozeduren gültig. Das Besondere daran ist, daß Sie hierbei Variablennamen angeben können, die auch im Hauptprogramm verwendet werden (sogenannte *globale Variablen*). Die Inhalte der globalen Variablen bleiben von den lokalen Variablen unbeeinflußt.

Seit der GFA-Basic-Version 3.0 ist es auch möglich, *mehrzeilige Funktionen* zu definieren. Sie kennen sicher die selbstdefinierten Funktionen (DEFFN...). Der Mangel dabei bestand darin, daß diese Funktion in nur einer Programmzeile definiert werden mußte. Waren umfangreichere Definitionen nötig, mußte auf eine Prozedur mit Übergabevariablen ausgewichen werden. Durch die Struktur

```
FUNCTION name
  ...
  Programmtext
  ...
  RETURN Wert
ENDFUNC
```

können Sie nun auch längere Funktionen definieren. Sie sehen hier auch gleich den Unterschied zu den Prozeduren: Bei Funktionen muß hinter RETURN ein Rückgabewert angegeben werden, der dem Hauptprogramm zurückgeliefert wird. Es ist erlaubt, mehrere RETURN-Befehle z.B. durch »IF Bedingung THEN RETURN Wert« innerhalb der Funktion zu nutzen. Die Funktion wird durch »ENDFUNC« beendet.

Eine Funktion kann mit

FN name oder
@ name

aufgerufen werden.

1.2.1 Übergabe von Parametern

Die Übergabe von Werten an Prozeduren und Funktionen ist natürlich das Salz in der Suppe bei der Verwendung von Unterprogrammen. Auf diesem Weg können Unterprogramme zu vielseitig verwendbaren, unabhängigen Hilfsroutinen werden. Auch sollten die in der Prozedur genutzten Variablen zu lokalen Variablen deklariert werden. Dadurch können eventuell vorhandene gleichnamige Variablen des Hauptprogramms nicht beeinflußt werden, was sonst zu häufigen unerklärlichen Fehlern führt.

Die Übergabe der Werte ist sehr einfach. Sie müssen dazu nur in der Titelzeile der Prozedur oder Funktion diese Variablen definieren. Dazu zwei Beispiele:

```
PROCEDURE kursiv_ausgabe(x,y,text$)
FUNCTION feld_mul(f1%(),f2%())
```

Es können beliebig viele Variablen sowie beliebige Variablentypen und auch Felder angegeben werden. Natürlich müssen diese beim Funktions- oder Prozeduraufruf mit Werten belegt werden.

```
@kursiv_ausgabe(160,100,"Probetext")
FN mul(alt%,neu%)
```

1.2.2 VAR-Parameter

Bei der Übergabe von Werten an Prozeduren und Funktionen werden jeweils die Unterprogramm-Variablen mit den Werten belegt. Eine andere Möglichkeit bietet sich mit den VAR-Parametern. Hierbei werden nämlich nicht nur die Werte an die Unterprogramme übergeben, sondern die Variablen selbst, die dann zu lokalen Variablen werden. Nach Rückkehr aus dem Unterprogramm erhalten die Übergabevariablen die neuen Werte. Dazu ein Beispiel:

```
REPEAT
 w1=RAND(100)
 w2=RAND(100)
 w3=RAND(100)
 abc(w1,w2,w3,w4,w5)
 PRINT AT(1,1);"Summe kleine Werte ";w4;" Summe große Werte ";w5
UNTIL 0
PROCEDURE abc(wert1,wert2,wert3,VAR wert4,wert5)
 ADD wert4,MIN(wert1,wert2,wert3)
 ADD wert5,MAX(wert1,wert2,wert3)
RETURN
```

In diesem Beispiel werden Zufallszahlen erzeugt. Jeweils die Werte der größten und der kleinsten Zahl werden in der Prozedur aufaddiert und als Variablen »w4« und »w5« ausgegeben. Diese Variablen werden in der Prozedur zu »wert4« und »wert5«. Es handelt sich aber um dieselben Variablen. Jede Änderung an »wert4« und »wert5« wird somit auch als Änderung an »w4« und »w5« bemerkbar sein.

1.3 Programmschleifen

Ein Hauptelement strukturiert aufgebauter Programme sind neben den Prozeduren und Funktionen die *Programmschleifen*. Die Programmschleifen führen das Programm wiederholt aus, solange eine Bedingung erfüllt oder nicht erfüllt ist. GFA-Basic 3.0 bietet Ihnen auf diesem Gebiet eine besonders große Auswahl an *Strukturbefehlen*, die auch schon als zu umfangreich kritisiert wurde. Wir möchten diese und die sogenannten Verzweigungsbefehle nun kurz auflisten.

DO LOOP

```
DO
  Programmtext
LOOP
```

Hierbei handelt es sich um eine Endlosschleife. Der in ihr enthaltene Programmtext wird also ständig wiederholt. Die Schleife kann jedoch durch den Befehl »EXIT IF Bedingung« verlassen werden, wenn die angegebene Bedingung erfüllt ist.

REPEAT UNTIL

```
REPEAT
  Programmtext
UNTIL Bedingung
```

Der Programmtext innerhalb dieser Schleife wird mindestens einmal ausgeführt und wiederholt, wenn die Bedingung nicht erfüllt ist.

WHILE WEND

```
WHILE Bedingung
  Programmtext
WEND
```

Hierbei wird schon zu Beginn der Schleife die Bedingung überprüft. Die Schleife wird ausgeführt und wiederholt, wenn die Bedingung erfüllt ist.

Die Zusätze WHILE und UNTIL für DO LOOP-Programmschleifen

Die Schleifenart DO LOOP kann in der Version 3.0 des GFA-Basic durch zwei Zusätze erweitert werden. Dabei bedeuten:

DO WHILE Bedingung

Die Schleife wird nur dann ausgeführt, wenn die Bedingung wahr ist.

DO UNTIL Bedingung

Hierbei wird die Schleife nur dann ausgeführt, wenn die Bedingung nicht erfüllt ist.

LOOP WHILE Bedingung

Wenn die Bedingung erfüllt ist, wird der Schleifeninhalt ab DO erneut abgearbeitet.

LOOP UNTIL Bedingung

Der Schleifeninhalt wird nur dann erneut abgearbeitet, wenn die Bedingung nicht erfüllt ist.

Diese Erweiterungen für die DO LOOP-Schleife können beliebig miteinander kombiniert werden.

FOR TO STEP NEXT

```
FOR variable = anfang TO ende [STEP stepwert]
  Programmtext
NEXT variable
```

Der Programmtext innerhalb dieser Programmschleife wird so oft wiederholt, bis der Wert »variable« größer ist als der »ende«-Wert. Dabei wird zu »variable« bei jedem Durchlauf 1 bzw. »stepwert« (kann auch negativ sein) addiert.

```
FOR variable = anfang DOWNTO ende
  Programmtext
NEXT variable
```

Die Verwendung von DOWNTO ist eine Alternative zu STEP-1. Dabei sollte man auf die richtigen Proportionen der Anfangs- und Endwerte achten, da sich sonst Endlosschleifen ergeben können.

Das Verlassen von Programmschleifen durch EXIT IF

EXIT IF Bedingung

Der Befehl kann in allen Schleifentypen beliebig oft eingesetzt werden. Dabei wird die momentan durchlaufene Schleife sofort verlassen, wenn die Bedingung erfüllt ist.

1.4 Entscheidungsanweisungen

Hierzu gehören IF THEN, SELECT CASE und ON GOSUB. Der Programmablauf wird jeweils von einer Bedingung oder einem Wert abhängig gemacht.

IF THEN

Die IF THEN-Struktur brauchen wir Ihnen sicher nicht mehr zu erläutern. GFA-Basic 3.0 kennt jedoch noch eine Erweiterung, die oftmals eine große Hilfe darstellt:

ELSE IF

```
IF Bedingung1 THEN
  Bedingung1 erfüllt
ELSE IF Bedingung2
  Bedingung2 erfüllt
ELSE IF Bedingung3
  Bedingung3 erfüllt
ELSE
  keine Bedingung erfüllt
ENDIF
```

An diesem kleinen Beispiel sehen Sie, wie ELSE IF wirkt. Es kann nacheinander auf mehreren Bedingungen geprüft werden. Eine vergleichbare Struktur hätte man mit mehreren IF THEN-Befehlen auch schreiben können. Diese wäre jedoch umfangreicher geworden, da jedesmal noch ein ENDIF zusätzlich in den Programmtext aufgenommen werden müßte. Außerdem können die sonst so übersichtlichen Einrückungen bei verschachtelten IF THEN-Strukturen auch unübersichtlich werden, wenn sehr viele

Bedingungen überprüft werden sollen. Ohne ELSE IF sieht das oben angeführte Beispiel folgendermaßen aus:

```
IF Bedingung1 THEN
 Bedingung1 erfüllt
ELSE
 IF Bedingung2 THEN
   Bedingung2 erfüllt
 ELSE
   IF Bedingung3 erfüllt
     Bedingung3 erfüllt
   ELSE
     keine Bedingung erfüllt
   ENDIF
 ENDIF
ENDIF
```

SELECT CASE

Ein neuer Verzweigungsbefehl steht Ihnen im GFA-Basic 3.0 mit der SELECT CASE-Anweisung zur Verfügung. Es handelt sich hierbei um eine ähnliche Struktur wie mit IF THEN ELSE IF, bietet jedoch darüber hinaus noch einige Besonderheiten.

```
SELECT Ausdruck
CASE Wert1
  Ausdruck entspricht Wert1
CASE Wert2
  Ausdruck entspricht Wert2
CASE Wert3
  Ausdruck entspricht Wert3
ENDSELECT
```

Hier werden die hinter CASE angegebenen Werte (Konstanten oder Variablen) mit dem Ausdruck verglichen. Ist das Ergebnis »wahr«, wird der Programmteil zwischen dem entsprechenden CASE und der folgenden Anweisung ausgeführt. Anschließend folgt der nächste Vergleich, falls vorhanden. Eine CASE-Anweisung kann durch Angabe eines besonderen Befehls (CONT) übersprungen werden.

```
SELECT Ausdruck
CASE Wert1
  Ausdruck entspricht Wert1
CASE Wert2
  Ausdruck entspricht Wert2
CONT
```

```
CASE Wert3
  Ausdruck entspricht Wert3
DEFAULT
Ausdruck entspricht keinem der Werte
ENDSELECT
```

In diesem Beispiel wird »CASE Wert3« übersprungen, wenn der zweite Vergleich zutrifft und CONT ausgeführt wird. Der Programmteil zwischen DEFAULT und ENDSELECT wird nur dann ausgeführt, wenn keiner der Vergleiche zutrifft.

Für die Schreibweise von CASE gibt es folgende Möglichkeiten:

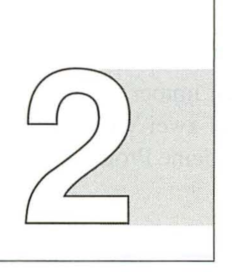
Schreibweise	**Beschreibung**	
CASE Wert	Ausdruck =	Wert
CASE Wert1 TO Wert2	Ausdruck =>	Wert1 und Ausdruck <= Wert2
CASE TO Wert2	Ausdruck <=	Wert2
CASE Wert1 TO	Ausdruck =>	Wert1

ON GOSUB

Ebenfalls von sehr großer Wichtigkeit bei der Programmstrukturierung ist die Mehrfachverzweigung mit ON GOSUB.

```
ON Ausdruck GOSUB Prozedur1,Prozedur2,Prozedur3 ...
```

Hierbei wird je nach Größe des Wertes die jeweilige Prozedur aufgerufen (Wert=1, Prozedur1; Wert=2, Prozedur2 ...). Bei der Entscheidung spielen mögliche Nachkommawerte des Ausdrucks keine Rolle – sie werden ignoriert.



Wissenswertes zur Programmierarbeit

In diesem Kapitel sollen einige Themen angesprochen werden, mit denen ein Programmierer immer wieder konfrontiert wird und die Ihre Programme professioneller machen. Sie erfahren bei unserem Streifzug einiges über die Speicherverwaltung, das Arbeiten mit mehreren Bildschirmen und wie man Musik im Hintergrund laufen lassen kann. Wir beginnen mit dem wichtigsten Thema, der Speicherverwaltung.

2.1 Die Speicherverwaltung

Für viele Anwendungen ist es erforderlich, Speicherplatz vor dem Zugriff des Basic zu schützen. In den geschützten *Bereich* lädt man dann zum Beispiel ».RSC«-Dateien oder unterhält in ihm mehrere Bildschirme. Wenn Sie GFA starten, fordert es vom Betriebssystem (es ist unter anderem auch für die Speicherverwaltung zuständig) fast den gesamten verfügbaren Speicher an. Soll ein Programm selbst Speicher reservieren, gibt es mehrere Möglichkeiten.

Erste Möglichkeit

Angenommen, man benötigt 100 Byte, auf die nur das Programm und nicht das Basic Zugriff haben soll. Wie Sie wissen, belegt jedes Zeichen in einer Stringvariablen ein Byte. Diese Tatsache machen wir uns zunutze und fügen folgende Programmzeilen in das Programm ein:

```
speicher$=SPACE$(100)
sp_adr%=VARPTR(speicher$)
```

In der Variablen »speicher$« stehen 100 Leerzeichen. Die Variable »sp_adr%« enthält nach dem Aufruf die Adresse des ersten Bytes. Man kann jetzt den so reservierten Speicher für eigene Zwecke nutzen. Einen gravierenden Nachteil hat die Sache aber: Man kann sich eigentlich nie sicher sein, daß die Adresse bei jedem Aufruf noch stimmt. Das hängt mit der *Garbage Collection* zusammen. Jedesmal, wenn das Programm eine neue Variable anlegt, überprüft der Basic-Interpreter, ob noch genügend Speicherplatz für das Anlegen vorhanden ist. Wenn nicht, räumt er alle nicht mehr benötigten Variablen samt Inhalt aus dem Speicher. Die noch aktuellen Variablen werden der Reihe nach verschoben. Mit dem Verschieben ändert sich auch deren Adresse im Speicher, die vorher gemerkte stimmt nicht

mehr. »Poked« man jetzt zum Beispiel einen Wert an die alte Adresse, passiert im günstigsten Fall gar nichts. Im schlimmsten Fall, und von ihm sollte man immer ausgehen, hat es den Absturz des Systems zur Folge. Selbst wenn Ihr Programm nur zwei Variablen benutzt, können Variablenleichen entstehen. Tippen Sie einmal folgendes kleine Programm ein:

```
FOR i%=0 TO 10
  a$=CHR$(i%)
  PRINT VARPTR(a$)
NEXT i%
```

Obwohl der Variablen »a$« lediglich ein neuer Inhalt zugewiesen wird, ändert sich die Adresse, ab der er zu finden ist, bei jedem Aufruf. Wir können die *Garbage Collection* auch gezielt auslösen. Als Beispiel dient wieder das obige Programm, es wurde lediglich um zwei Zeilen ergänzt.

```
FOR i%=0 TO 10
  a$=CHR$(i%)
  PRINT VARPTR(a$)
NEXT i%
PRINT FRE(0)
PRINT VARPTR(a$)
```

Das Aufräumen des Speichers veranlaßt der Befehl »PRINT FRE(0)«. Als neue Adresse erhalten Sie die erste, vom Programm ausgegebene, Adresse zurück. Bei Verwendung dieser Reservierungsmethode übergeben Sie am besten die Adresse immer mit »VARPTR(speicher$)+offset«. Mit diesem Verfahren lassen sich 32767 Byte für eigene Anwendungen reservieren.

Zweite Möglichkeit

Wie schon erwähnt, fordert GFA beim Start fast den gesamten Speicherplatz vom Betriebssystem an. Damit ein Programm selbst Speicher anfordern kann, muß zuerst einmal Speicher an das Betriebssystem zurückgegeben werden. Dazu dient der Befehl:

RESERVE(Anzahl)

In der Variablen »Anzahl« steht die Speichergröße, die dem Basic (für Variablen etc.) nach dem Aufruf noch zur Verfügung stehen soll. Es muß sich dabei um eine durch 256 teilbare Zahl handeln. Beispiel: »RESERVE(512)« reserviert für Variablen und Felder 512 Byte, der Rest wird wieder dem Betriebssystem zur Verfügung gestellt. Dieser Aufruf kommt immer dann in Frage, wenn der danach zur Verfügung gestellte Speicher auch wirklich gebraucht wird. Oft braucht man aber nur einen kleinen Teil des Speichers. In »Anzahl« steht dann zweckmäßigerweise ein negativer Wert. Beispiel: »RESERVE(–512)«. GFA gibt 512 Byte an das Betriebssystem zurück. Zum Herstellen des ursprünglichen Zustands ruft man »RESERVE« ohne Parameter auf.

Die Rückgabe von Speicher an das GEMDOS (dieser Teil des Betriebssystems ist nämlich für die Speicherverwaltung zuständig) reicht noch nicht. Das Programm muß ihn nämlich erst einmal wieder anfordern. Das geht mit folgendem Befehl:

```
Adresse%=MALLOC(Anzahl)
```

In »Anzahl« übergibt man die Anzahl der benötigten Bytes. Übergabe von –1, liefert die Länge des größten zusammenhängenden Speicherbereichs zurück. In »Adresse%« steht, nach Aufruf mit positivem Wert, die Anfangsadresse des reservierten Bereichs. Trat ein Fehler auf, ist »Adresse%« gleich null. Am Ende des Programms muß es den angeforderten Speicherplatz wieder zurückgeben. Die interne Speicherverwaltung des GFA-Interpreters gerät sonst nachhaltig durcheinander. Die Rückgabe erfolgt über den Befehl:

```
MFREE(ADRESSE%)
```

In »Adresse%« übergibt man die beim »MALLOC« erhaltene Adresse. Damit das Programm ordentlich abgeschlossen wird, folgt noch der Befehl »RESERVE«. Diesmal ohne Parameter.

Hinweis: Wenn das Programm mehrere Speicherbereiche benötigt, sollte man diese nicht nacheinander über »MALLOC« reservieren lassen. Ältere Betriebssysteme des ST weisen nämlich einen Fehler bei der Speicherverwaltung auf. Es sollen wohl nur ca. 80 MALLOC-Aufrufe, nach jedem Neustart des Rechners, fehlerfrei möglich sein. Reservieren Sie darum einen großen Speicherblock, und schreiben sich dann die Verwaltung des Blocks in Basic selbst.

Das folgende Programm zeigt Ihnen, was man zum Beispiel mit einem reservierten Speicherbereich anfangen kann. Es hat den Namen 2_1.GFA. Das Programm stellt Ihnen, entsprechend dem Speicherausbau Ihres ST, mehrere Bildschirme zur Verfügung.

2.2 Das Arbeiten mit mehreren Bildschirmen

Vielleicht haben auch Sie schon Programme kennengelernt, bei denen in Null komma nichts mehrere ziemlich komplexe Grafiken nacheinander auf dem Bildschirm erscheinen. Der Trick liegt dabei nicht unbedingt in ihrer schnellen Berechnung, sondern meistens baut der Rechner das folgende Bild im Hintergrund auf, während er ein anderes noch anzeigt. Wie das geht, zeigt Ihnen das folgende Programm. Zuerst müssen wir Ihnen allerdings noch etwas über die Lage des Bildschirms im Speicher erklären.

Der Bildschirm ist beim ST in zwei Speicherbereiche aufgeteilt. Jeder Speicherbereich besteht aus 32.000 Byte, jedes Bit entspricht dabei (beim monochromen Monitor) einem Punkt des Bildschirms. Die Teile bestehen aus dem sichtbaren Teil (der entsprechende Speicherinhalt wird dargestellt) und einem »unsichtbaren« Teil, in ihm werden die Bildschirmausgaben vorgenommen. Die Anfangsadresse des sichtbaren Teils nennt man *Physbase*, die Anfangsadresse für den »unsichtbaren« *Logbase*. Damit Bildschirmausgaben

sofort dargestellt werden, sind nach dem Einschalten des Rechners und beim Arbeiten mit GFA zunächst beide Adressen gleich. Das hat noch einen weiteren Vorteil: Die 32.000 Byte werden nur einmal und nicht zweimal benötigt.

Das Programm »splittet« nun die Teile auf, d.h. die Adresse für die Physbase ist eine andere als die für die Logbase. Die jeweiligen Adressen erhält man über den Aufruf des XBIOS, ein Teil des Betriebssystems. Nach dem Aufruf von:

Physbase%=XBIOS(2)

steht in der Variablen »Physbase%« die Anfangsadresse des sichtbaren Bildschirmteils. Die dem XBIOS-Aufruf in Klammern gesetzte Zahl ist die entsprechende Funktionsnummer des XBIOS.

Die Anfangsadresse für den Ausgabebildschirm erhält man über den folgenden Aufruf:

Logbase%=XBIOS(3)

Es gilt sinngemäß das oben Gesagte. Für das Verändern beider oder auch nur einer Adresse(n) und der Auflösung ist der nächste XBIOS-Aufruf zuständig:

~**XBIOS**(5,L:Log_base%,L:Phys_base%,Aufl)

Die Adressen müssen als Langwort übergeben werden, darum die jeder Adresse vorangestellten Zeichen »L:«. Die Variablen »Log_base%« und »Phys_base%« müssen jeweils eine durch 256 teilbare Adresse enthalten. Übergabe von –1 bedeutet, daß der jeweilige Wert nicht verändert wird.

Für die Auflösung gilt:

0 = niedrige,
1 = mittlere,
2 = hohe Auflösung

Das reicht vorerst, es folgt das Programm. Nach dem Start und einer kurzen Pause können Sie über die [+]- und [-]-Taste zwischen verschiedenen Bildschirmen umschalten, wie viele es sind, hängt vom Speicherausbau Ihres Rechners ab. Das Programm wird mit der [Esc]-Taste beendet.

```
ON ERROR GOSUB speicher_zurueck          !Wenn Fehler
ON BREAK GOSUB ende                      !Wenn Stop
'
```

Da unser Programm über »RESERVE« und »MALLOC« Speicher für sich reserviert und dieser immer ordnungsgemäß zurückgegeben werden muß, beginnt unser Programm mit obigen Zeilen.

```
RESERVE 1024                            !1024 Byte für das Programm
max%=MALLOC(-1)                         !Wieviel Speicher ist übrig?
adr%=MALLOC(max%)                       !Speicher reservieren
```

Für die Variablen soll das Basic lediglich 1024 Byte behalten. Der Rest wird an das Betriebssystem zurückgegeben. Das Programm erfragt danach den größten zusammenhängenden Speicherblock beim GEMDOS. Die Größe steht dann in der Variablen »max%«. Der nächste Aufruf reserviert den Speicherblock für das Programm. Seine Anfangsadresse steht danach in der Variablen »adr%«.

```
bs_anz&=(max%-256)/2^15                 !Wie viele Bildschirme sind möglich?
ALERT 1,STR$(bs_anz&+1)+" BILDSCHIRME| |SIND MÖGLICH",1," OK ",bu&
DIM bs_adr%(bs_anz&)                    !Entsprechend dimensionieren
```

Das Programm berechnet die Anzahl der möglichen Bildschirme und zeigt Sie Ihnen an. In dem dann entsprechend dimensionierten Feld stehen später die Anfangsadressen der Bildschirme.

```
anf_adr%=((adr%+256)*256)/256           !Erste Adresse berechnen
bs_adr%(0)=XBIOS(2)                     !Aktueller Bildschirm
CLS
PRINT "BILDSCHIRM 1"
```

Die Variable »anf_adr%« enthält nach der Berechnung die Anfangsadresse des ersten Bildschirms. Auch der aktuelle Ausgabebildschirm soll mit in das Programm einbezogen werden. Seine Adresse erfahren wir über die schon besprochene XBIOS-Funktion. Danach wird er gelöscht und erhält seinen Inhalt.

```
FOR i&=1 TO bs_anz&                     !Bilddschirme anlegen
  bs_adr%(i&)=anf_adr%+(i&-1)*2^15      !Adresse des Bildschirms
  ~XBIOS(5,L:bs_adr%(i&),-1,-1)         !Logbase umschalten
  CLS                                   !Bildschirm löschen
  PRINT SPACE$(i&*4)+"BILDSCHIRM ";i&+1 !Bildinhalt
NEXT i&
CLR i&
~XBIOS(5,L:bs_adr%(i&),L:bs_adr%(i&),-1)!Erster Bildschirm ein
```

Die anderen Bildschirme richtet das Programm in einer Schleife ein. Nach der Adreßberechnung schaltet das Programm den Ausgabebildschirm um, löscht ihn und baut eine Information in ihm auf. Sie, als Anwender, bekommen davon nichts mit, das ist ja das Schöne. Am Ende der Schleife erhalten Logbase und Physbase wieder die gleiche Adresse, sprich: alle Änderungen des Bildschirminhalts sind wieder sofort sichtbar.

```
DO
  SELECT INP(2)                               !Auf Taste warten
  CASE 43                                     !Plus-Taste
    i&=MIN(i&+1,bs_anz&)
    ~XBIOS(5,L:bs_adr%(i&),L:bs_adr%(i&),-1)!Log-& Physbase ändern
  CASE 45                                     !Minus-Tasten
    i&=MAX(i&-1,0)
    ~XBIOS(5,L:bs_adr%(i&),L:bs_adr%(i&),-1)!Log-& Physbase ändern  CASE 27
!ESC-Taste
    GOSUB ende
  ENDSELECT
LOOP
```

Die Schleife übernimmt die Abfrage der Tastatur und schaltet entsprechend auf den vorherigen oder folgenden Bildschirm um.

```
PROCEDURE ende
  ~XBIOS(5,L:bs_adr%(0),L:bs_adr%(0),-1)     !Ur-Bildschirm ein
  ~MFREE(adr%)                                !Speicher an Betriebssystem
  RESERVE                                     !Speicher zurück an GFA
  END
RETURN
```

Am Ende des Programms schaltet dieses erst einmal wieder den »Ur-Bildschirm« ein. Die nächste Zeile gibt den reservierten Speicher wieder an die GEMDOS-Verwaltung zurück. »RESERVE« schließlich stellt die ursprüngliche Speichergröße des GFA-Basic wieder her. Der nächste Abschnitt behandelt die Musikuntermalung Ihrer Programme.

2.3 Geräusche und Musik ohne Programmunterbrechung

Im ST ist ein Chip eingebaut, der Musik und Geräusche erzeugen kann (er hat zwar auch noch andere Aufgaben zu erledigen, sie interessieren an dieser Stelle jedoch nicht). Das GFA-Basic kennt daher auch Befehle, die den Programmierer schnell in die Lage versetzen, eigene Liedchen oder Geräusche zu programmieren und abspielen zu lassen. Einen Nachteil haben sie: Während ihrer Abarbeitung »steht« das Programm, es erzeugt also nur Klänge. Im Betriebssystem gibt es aber eine Funktion, die uns von diesem Manko befreit. Sie heißt: »Dosound« und wird wie folgt angesprochen:

~**XBIOS**(32,L:Adresse%)

In der Variablen »Adresse%« übergibt man der Funktion die Anfangsadresse eines vorher reservierten Bereichs, ab der die Daten stehen, die das Musikstück näher beschreiben. Übergibt man statt der Adresse –1, erhält man den aktuellen Stand der Bearbeitung zurück. Der Wert Null bedeutet dabei, daß zur Zeit kein Sound gespielt wird. Dazu später mehr.

Mit dem Aufruf (mit Adresse) veranlassen wir das Betriebssystem, unsere Musik während des Systeminterrupts zu bearbeiten. Vom Interrupt bekommt der Anwender nichts mit, er wird sehr schnell abgearbeitet. Interrupt heißt, daß das laufende Programm unterbrochen wird, damit der Rechner noch andere wichtige Dinge erledigen kann (zum Beispiel Abfrage der Tasten [Alt]+[Help]), Erhöhen der Systemzeit). Diese Interrupt-Routine überprüft nun auch, ob »Dosound« aktiv ist. Wenn ja, wird sie auch noch schnell mit abgearbeitet. Bevor wir mit der Musikerstellung beginnen, wichtige Grundlagen zum Musikchip:

Der Chip besitzt drei unabhängige Tongeneratoren und einen Rauschgenerator. Jede Stimme kann mit dem Rauschgenerator gemischt werden, man erhält so interessante Klänge. Einstellen lassen sich auch noch die Hüllkurve und die Lautstärke. Damit der Chip weiß, welchen Ton er von sich geben soll, verfügt er über mehrere Register. Sie werden bei unserem Beispiel zwar von der »Dosound«-Routine belegt, die Belegungswerte jedoch stammen vom Programmierer.

Register	Funktion
0	Es ermöglicht die Feineinstellung der Frequenz des Kanals A.
1	Die unteren 4 Bit stellen die Frequenz des Kanals A grob ein.
2	Kanal B, Funktion wie Register 0.
3	Kanal B, Funktion wie Register 1.
4	Kanal C, Funktion wie Register 0.
5	Kanal C, Funktion wie Register 1.
6	Dient zur Einstellung der Rauschfrequenz. Es werden nur die Bits 0–4 verwendet. Je kleiner der Wert, um so höher das Rauschen.
7	Es werden nur die unteren 6 Bit beschrieben: Bit 0: 0 = Kanal A ein, 1 = Kanal A aus. Bit 1: 0 = Kanal B ein, 1 = Kanal B aus. Bit 2: 0 = Kanal C ein, 1 = Kanal C aus. Bit 3: 0 = Rauschen + Kanal A, 1 = Rauschen Kanal A aus. Bit 4: 0 = Rauschen + Kanal B, 1 = Rauschen Kanal B aus. Bit 5: 0 = Rauschen + Kanal C, 1 = Rauschen Kanal B aus.
8	Die Lautstärke des Kanals A wird mit den Bits 0–3 beeinflußt. Ist Bit 4 gesetzt, wird die Lautstärke durch die Hüllkurve bestimmt. Der Inhalt von Bit 0 bis Bit 3 ist dann belanglos.
9	Lautstärke des Kanals B, wie Register 8.
10	Lautstärke des Kanals C, wie Register 8.

Register	Funktion
11	Feineinstellung der Hüllkurvenfrequenz.
12	Grobeinstellung der Hüllkurvenfrequenz.
13	Die Bits 0–3 bestimmen die Hüllkurve, die Zuordnung entspricht der des GFA-Basic.

Der Chip kennt einige Befehle. Sie werden wie die Noten übergeben, sprich, sie stehen ebenfalls im Speicher. Jeder Befehl ist ein Byte lang. Ihm müssen evtl. noch mehr Bytes folgen, die ihn näher beschreiben.

Befehlscode	Funktion
$00–$0F	Jeder Code wird als eine Registernummer unseres Soundchips interpretiert. Das einem dieser Befehle folgende Byte wird in das entsprechende Register geladen. Für die Musikprogrammierung sind nur die Codes $00 bis $0D interessant.
$80	Dieser Befehl lädt das ihm folgende Byte in ein vorläufiges Register.
$81	Dem Befehl folgen drei Byte. Das erste bestimmt, in welches Soundchipregister der Wert des vorläufigen Registers geladen wird. Das zweite ist ein Wert in zweierkomplementärer Form und wird zum Inhalt des vorläufigen Registers addiert. Das dritte Byte schließlich bestimmt das Tonende. Es ist erreicht, wenn das vorläufige Register und das Byte für das Tonende den gleichen Wert haben.
$82–$FF	Auch diesem Befehl muß ein Byte folgen. Ist es null, wird die Abarbeitung beendet, andere Werte bestimmen die Tonlänge (in 20-ms-Schritten).

Das Wichtigste haben wir nun erklärt. Es fehlt Ihnen aber noch eine Notentabelle, damit das, was Sie programmieren möchten, auch ansprechend klingt. Wir haben sie mit freundlicher Genehmigung dem Buch »Programmierung von Grafik und Sound auf dem Atari ST« (erschienen im Markt&Technik-Verlag) entnommen.

Note	Frequenz	Periodenwert	H-Byte	L-Byte
C1	32.703	3822	14	238
Cis	34.648	3608	14	24
D1	36.708	3405	13	77
Dis	38.891	3214	12	142
E1	41.203	3034	11	218
F1	43.654	2863	11	47
Dis1	46.249	2703	10	143
G1	48.99	2551	9	247
Gis1	51.913	2408	9	104
A1	55.000	2273	8	225
B1	58.270	2145	8	97
H1	61.735	2025	7	233
C	65.406	1911	7	119
Cis	69.296	1804	7	12
D	73.416	1703	6	167
Dis	77.782	1607	6	71
E	82.407	1517	5	237
F	87.307	1432	5	152
Fis	92.499	1351	5	71
G	97.999	1276	4	252
Gis	103.826	1204	4	180
A	110.000	1136	4	112
B	116.541	1073	4	49
H	123.471	1012	3	244
c	130.813	956	3	188
cis	138.591	902	3	134
d	146.823	851	3	83
dis	155.564	804	3	36
e	164.814	758	2	246
f	174.614	716	2	204
fis	184.997	676	2	164
g	195.998	638	2	126
gis	207.552	602	2	90
a	220.000	568	2	56
b	233.082	536	2	24
h	246.942	506	1	250
c1	261.626	478	1	222
cis1	277.183	451	1	195
d1	293.665	426	1	170
dis1	311.127	402	1	146
e1	329.628	379	1	123

Note	Frequenz	Periodenwert	H-Byte	L-Byte
f1	349.228	358	1	102
fis1	369.994	338	1	82
g1	391.995	319	1	63
gis1	415.305	301	1	45
a1	440.000	284	1	28
b1	466.164	268	1	12
h1	493.883	253	0	253
c2	523.251	239	0	239
cis2	554.365	225	0	225
d2	587.330	213	0	213
dis2	622.254	201	0	201
e2	659.255	190	0	190
f2	698.457	179	0	179
fis2	739.989	169	0	169
g2	783.991	159	0	159
gis2	830.609	150	0	150
a2	880.000	142	0	142
b2	932.328	134	0	134
h2	987.767	127	0	127
c3	1046.502	119	0	119
cis3	1108.731	113	0	113
d3	1174.659	106	0	106
dis3	1244.508	100	0	100
e3	1318.510	95	0	95
f3	1396.913	89	0	89
fis3	1479.978	84	0	84
g3	1567.982	80	0	80
gis3	1661.219	75	0	75
a3	1760.000	71	0	71
b3	1864.655	67	0	67
h3	1975.533	63	0	63
c4	2093.004	60	0	60
cis4	2217.461	56	0	56
d4	2349.318	53	0	53
dis4	2489.016	50	0	50
e4	2637.021	47	0	47
f4	2793.826	45	0	45
fis4	2959.955	42	0	42
g4	3135.963	40	0	40
gis4	3322.438	38	0	38
a4	3520.000	36	0	36

Note	Frequenz	Periodenwert	H-Byte	L-Byte
b4	3729.310	34	0	34
h4	3951.066	32	0	32
c5	4186.009	30	0	30
cis5	4434.922	28	0	28
d5	4698.636	27	0	27
dis5	4978.032	25	0	25
e5	5274.041	24	0	24
f5	5587.652	22	0	22
fis5	5919.911	21	0	21
g5	6271.927	20	0	20
gis5	6644.875	19	0	19
a5	7040.000	18	0	18
b5	7458.621	17	0	17
h5	7902.133	16	0	16

Auf der Diskette finden Sie das Programm 2_2.GFA. Laden Sie es jetzt bitte einmal ein und starten Sie es anschließend. Es erscheint eine Balkengrafik. Die kleinen Balken zeigen Ihnen die Länge des vorherigen Tons bzw. der vorherigen Pause. Der rechte Balken stellt die aktuelle Länge dar. Die Geschwindigkeit, in der die Grafik aufgebaut wird, spricht einmal für das GFA-Basic, zum anderen für diese Art der Musikprogrammierung. Das Programm braucht sich nur einmal, am Anfang, um die Musik zu kümmern. Den Rest erledigt das Betriebssystem.

```
' ************************** BEMU 89 ***************************
' ********************** Musik im Interrupt *******************
' ************** Programm aus dem GfA 3.0 Buch ***************
' ************ Von Wilhelm Besenthal & Jens Muus *************
' ********* erschienen im Markt & Technik Verlag 1989 ********
' *************************************************************
'
a$=SPACE$(1000)                 !Speicher für Noten
DEFFILL 1,2,4                   !Hintergrundmuster
BOUNDARY 0                      !Umrahmung aus
```

Wir reservieren in der Variablen »a$« Speicherplatz für die Noten. Wir gehen davon aus, daß das Basic keine Garbage Collection vornimmt. Für Ihre eigenen Programme empfehlen wir Ihnen, Speicher über RESERVE und MALLOC zu reservieren.

```
FOR i&=1 TO 16                  !Füllmuster lesen
  READ f&
  fuell$=fuell$+MKI$(f&)
NEXT i&
```

Die Schleife liest ein selbstdefiniertes Füllmuster in die Variable »fuell$«. Der nächste Schritt besteht im Übertragen der Werte aus den DATA-Zeilen in den reservierten Speicherbereich. Sie können auch mit dem INLINE-Befehl arbeiten, seine Beschreibung folgt im Kapitel 7.

```
DO                              !Noten lesen
  READ wert%
  EXIT IF wert%=-1
  POKE VARPTR(a$)+i%,wert%
  INC i%
LOOP
```

Das Übertragen der Daten in den Puffer übernimmt die obige Schleife.

```
DO
  WHILE XBIOS(32,-1)<>0             !Noch Noten da?
    DEFFILL 1,fuell$                !Definiertes Füll-Muster
    s%=XBIOS(32,-1)                 !Stand des Zählers holen
    REPEAT
      INC t&                        !Timer hochzählen
      PBOX 607,399-t&,639,399       !Box mit entspr. Höhe zeichnen
    UNTIL s%<>XBIOS(32,-1)          !Immer noch gleicher Zählerstand?
    PBOX p&,399-t&,p&+5,399         !Sonst Länge des Tons zeichnen
    DEFFILL 1,2,4                   !Hintergrundmuster
    PBOX 607,0,639,399              !Rechte Box löschen
    CLR t&                          !Timer löschen
    ADD p&,5                        !X-Position +5
  WEND
  CLR p&                            !X-Position löschen
  PBOX 0,0,639,399                  !Hintergrund löschen
  ~XBIOS(32,L:VARPTR(a$))           !DOSOUND aufrufen
LOOP
```

Das Hauptprogramm besteht aus drei Schleifen. Bei der äußeren handelt es sich um eine Endlosschleife. Das Programm läuft so lange, bis Sie [Control]+[Shift]+[Alternate] drücken. Die zweite Schleife sorgt dafür, daß, wenn die Musikstücke zu Ende sind, der Bildschirm gelöscht und die Musik erneut gespielt wird. Die dritte, innerste Schleife, übernimmt durch ständiges Abfragen des Dosound-Zeigers die Größenbestimmung der Balkendiagramme.

Wer nicht glaubt, daß die Musik unabhängig vom GFA-Basic erklingt, der sollte das Programm starten und nach den ersten Tönen mit [Control]+[Shift]+[Alternate] abbrechen. Anschließend ist in der »Ende-Box« OK anzuklicken. Der Editor erscheint wieder, die Musik spielt weiter. Wichtig ist, keine Taste zu drücken, da der Tastaturklick ebenfalls im Systeminterrupt erzeugt wird. Vor einem Tastaturklick wird die Musikverarbeitung abge-

brochen und auch später nicht wieder aufgenommen. Sie können den Klick aber über das Kontrollfeld abschalten.

Damit Sie nicht mit den gleichen Schwierigkeiten zu kämpfen haben wie wir, erläutern wir Ihnen einige Data-Zeilen.

```
' FÜLLMUSTER
'
DATA 0,15420,8736,8736,15416,8736,8736,15420
DATA 0,8740,13860,10788,10788,8740,8764,0
'
```

Diese Datas beschreiben das Füllmuster. Jedes gesetzte Bit entspricht auch einem gesetzten Bit im Füllmuster.

```
' LIEDER-DATAS
'
' I come from Alabama ......
'
DATA 7,252,13,0,12,90,11,0,8,16,9,16
```

Die Kanäle A und B werden eingeschaltet (7,252), die Hüllkurve gesetzt (13,0), die Frequenz der Hüllkurve bestimmt (12,90,11,0) und für Kanal A und B die maximale Lautstärke eingestellt (8,16,9,16).

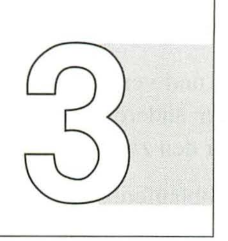
```
DATA 1,3,0,83,3,3,2,77,130,5,13,0
```

Die Frequenz des Kanals A (1,3,0,83), das entspricht der Note »d«, und des Kanals B (3,3,2,77), das entspricht einer Note zwischen »d« und »dis«, wird eingestellt. Durch das dichte Beieinanderliegen der Töne wird ein wesentlich vollerer Klang erreicht. Vor dem Spielen des nächsten Tons wird eine Pause von 5*20 ms eingelegt (130,5). Die Hüllkurve muß nach jedem Takt neu gesetzt werden, das übernehmen die Datas 13,0.

```
DATA 1,2,0,246,3,2,2,240,130,5,13,0
```

Kanal A spielt als nächstes eine Note »e«, Kanal B eine, die zwischen »e« und »f« liegt. Die Tondauer beträgt wieder 5*20 ms (130,5). Den Schluß der Zeile bildet erneut das Neusetzen der Hüllkurve.

Das soll reichen. Wir wünschen ihnen viel Spaß bei der Musikuntermalung eigener Programme.



Die GEM-Programmierung

Mit der Einführung von grafischen Benutzeroberflächen hat bei der Bedienung von Computern ein neues Zeitalter begonnen. Die wichtigsten Funktionen werden über die Maus angewählt, Dialogboxen und Drop-down-Menüs machen Untermenüs fast überflüssig. Auch der ST verfügt über diese sinnvolle Einrichtung. In der ersten Zeit seines Daseins gab es noch viele Programmierer, die Menüs nach der alten Methode (Eingabe über die Tastatur) erzeugten. In letzter Zeit hat sich das Bild grundlegend geändert. Die meisten aller aufwendigeren Programme lassen sich heute über die Maus bedienen und verfügen über die oben genannten Features. Die Beurteilung der Leistungsfähigkeit einer Programmiersprache für den ST hängt nicht zuletzt davon ab, wieweit sie den Programmierer bei dieser Art der Programmierung unterstützt. In diesem Punkt hat das GFA-Basic auf jeden Fall eine Bestnote verdient, Sie werden das im folgenden schnell selbst feststellen können.

Die Steuerung eines Rechners erfolgt über sein *Betriebssystem*. Im Atari heißt es TOS. Je nach Ausführung des Rechners wird es nach dem Einschalten von der Diskette geladen oder ist in ROMs (nicht löschbaren Speicherbausteinen) im Rechner fest eingebaut. Seine Größe beträgt stolze 196 Kbyte. Das Betriebssystem gliedert sich in mehrere Teile und verschiedene Hierarchien.

Jeder Teil hat seinen eigenen Namen. Die unterste Ebene stellen das BIOS (*Basic Input Output System*) und das XBIOS (*Extended Basic Input Output System*) dar. Beide Teile sind hardwareabhängig und ermöglichen z.B. die Kommunikation mit dem Diskettenlaufwerk. Den hardwareunabhängigen Teil der untersten Ebene stellt das GEMDOS (*Graphic Environment Manager Disk Operating System*) dar, es verwaltet zum Beispiel den Speicher. Über den besprochenen Teilen ist das GEM angesiedelt und verfügt über die Funktionen, die das Arbeiten mit dem ST zur Freude werden lassen. Es bedient sich dabei der Funktionen der unteren Ebene. GEM besteht aus zwei großen Teilen, dem VDI und dem AES. Das AES übernimmt die Kommunikation mit dem Benutzer und bedient sich dabei oft des VDI. Dieser Teil beinhaltet z.B. Grafikroutinen.

Dieses Kapitel behandelt das Aufrufen des AES. Nach dem Lesen werden Sie in der Lage sein, Programme zu schreiben, die Drop-down-Menüs und Dialogboxen benutzen. Beginnen wir mit der Erstellung und Verwaltung eines Drop-down-Menüs.

3.1 Die Drop-down-Menüs

Diese Menüart läßt sich unter GFA 3.0 auf zwei Wegen erstellen und verwalten. Der eine Weg führt über den Aufruf des AES über C-ähnliche Befehle, der andere über die GFA-eigenen Befehle. Beide Wege führen zum Ziel, wir beschreiben nur den zweiten.

Ein Programm, das über eine Menüleiste verfügt, sollte in etwa so ablaufen:

- Dimensionierung eines Feldes mit der Anzahl der Menütitel und Menüeinträge.
- Belegen des Feldes mit den Titeln und Einträgen.
- Bildschirm aufbauen.
- Übergabe des Feldes an GFA, das dann die Menüleiste im Speicher anlegt und darstellt.
- Warten auf Aktionen.
- Erfragen des gewählten Menüpunktes und entsprechende Reaktion. Danach den Titel des Menüs wieder normal darstellen lassen.
- Zurück zum vorletzten Schritt.

Damit Sie nicht lange rätseln müssen, wie man die einzelnen Programmpunkte programmiert, haben wir ein Demo-Programm für Sie geschrieben. Sie finden es unter dem Namen 3_1.GFA auf der beiliegenden Diskette. Anhand des Listings erklären wir nicht nur die Verwaltung der Menüleiste, sondern auch noch andere, sehr nützliche Funktionen des AES.

Nach dem Start sollte Ihr Bildschirm ungefähr so aussehen:

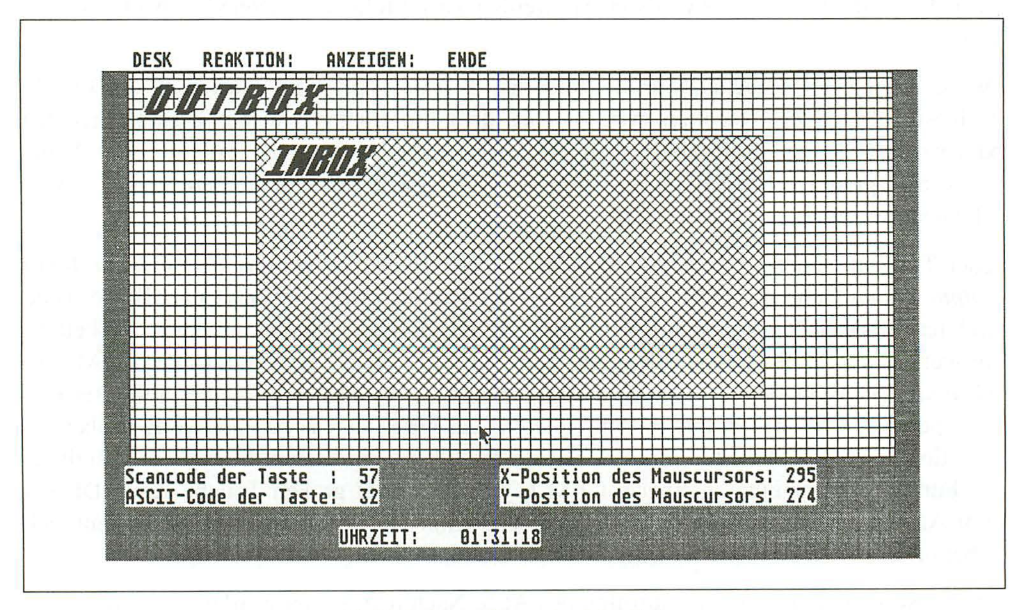


Bild 3.1: Verwaltung der Menüleiste

Zum Listing:

```
GOSUB menue_leiste_erstellen
GOSUB bild_aufbau
GOSUB warten
'
```

Das Programm besteht nur aus Unterprogrammen, die nacheinander aufgerufen werden.

```
PROCEDURE menue_leiste_erstellen
  ON MENU GOSUB menue_auswertung
  DIM eintrag$(26)
  DO                                    !Menüeinträge ins Array einlesen
    READ eintrag$(i&)
    EXIT IF eintrag$(i&)="-1-1"
    INC i&
  LOOP
  '
  DATA DESK , ÜBER DAS PROGRAMM ,---------------,1,2,3,4,5,6,""
  DATA REAKTION: ,  OUTBOX AUS ,  INBOX  AUS ,--------------
  DATA   TIMER EIN ,  TIMER AUS ,""
  DATA ANZEIGEN: ,  POSITION ,  TASTATUR ,""
  DATA ENDE , ENDE ,""
  '
  DATA "",""
  DATA -1-1
RETURN
```

Die Menüeinträge stehen später im Feld »Eintrag$« und werden über die DO...LOOP-Schleife in das Feld eingelesen. Der Aufbau des Stringarrays muß nach folgenden Regeln erfolgen:

Der erste Eintrag gibt den Titel des ersten Menüs vor. Unter diesem Menü sind später die beim Booten des Rechners eingeladenen Accessories zu finden. Der nächste ist der erste Eintrag des ersten Menüs, dessen Anwahl später meist eine Copyright-Meldung auf den Bildschirm bringt. Als nächstes folgt eine Anzahl von Minuszeichen (–), sie trennen den ersten Eintrag von dem nächsten und müssen ebenfalls immer vorhanden sein. Die nächsten sechs Einträge bestehen aus *Dummy-Strings*. Sie dienen als Platzhalter für die Accessory-Einträge. Den Abschluß bildet ein Leerstring, erzeugt durch zweimal Anführungszeichen oben (""). Der nächste String ist dann der Titel des zweiten Drop-down-Menüs. Alle ihm bis zum nächsten Leerstring folgenden Strings stellen die Einträge dar, die unter ihm zu finden sind. Den Abschluß der gesamten Einträge bildet ein weiterer Leerstring, d.h., es befinden sich zwei Leerstrings in Folge im Array. Der Aufbau eines Menüs mit drei Titel sieht also so aus:

```
Titel 1..Copyright..Dummy 1 bis Dummy 6..Leerstring
Titel 2..Eintrag1..Eintrag2..........Leerstring
Titel 3..Eintrag1..Eintrag2..........Leerstring..Leerstring
```

Achtung: Einen Fehler hat GFA 3.0 vom GFA 2.xx auf jeden Fall übernommen: Unter dem letzten Titel einer Menüleiste muß sich mindestens *ein* Menüeintrag verbergen. Wenn nicht, kommt es zum Absturz des Programms, manchmal auch des gesamten Systems.

Sollen später Einträge durch *Check-Marks* gekennzeichnet werden, sollte den Menüeinträgen mindestens ein Leerzeichen vorangestellt sein. Das Häkchen stünde sonst auf dem ersten Zeichen des Eintrags. Es geht mit dem Listing weiter.

Den ersten Befehl des obenstehenden Unterprogramms beschreiben wir etwas später im Zusammenhang mit anderen, gleichartigen. Das nächste Unterprogramm übernimmt den Bildschirmaufbau.

```
PROCEDURE bild_aufbau
  DEFTEXT 1,13,0,26
  DEFFILL 1,2,4
  BOUNDARY 0                        !Umrahmung ausschalten
  PBOX 0,0,640,400                  !Bildschirmhintergrund
  OPENW 0
  MENU eintrag$()
  MENU 15,2
  '
```

Für die Schriftzüge im Bild benutzen wir Fettschrift mit einer Pixelhöhe von 26. Das Füllmuster wird als das bekannte vom Desktop definiert. Das Programm schaltet danach die Umrandung von Flächen aus, der Befehl »PBOX 0,0,640,400« erstellt den Hintergrund.

OPENW 0

Der vorstehende Befehl erzeugt einen weißen Balken am oberen Bildschirmrand, der späteren Position unserer Menüleiste. Er sollte immer eingesetzt werden, da er das Überschreiben der Menüleiste durch Ausgabe- und Grafikbefehle verhindert. Nach seinem Aufruf werden Y-Positionen innerhalb der Menüleiste als negative Werte zurückgegeben. Das Programm kann so schnell entscheiden, welche Aktionen es zulassen möchte oder nicht. Der neue Ursprung des Bildschirms liegt, nach dem Aufruf, 19 Punkte tiefer als der alte.

Der nächste Befehl übergibt GFA 3.0 den Namen des Feldes, in dem die Einträge der Leiste zu finden sind. GFA erzeugt danach die Menüleiste im Speicher, die Namen der Einträge sind deshalb später nochmals, an anderer Stelle des Speichers, zu finden. Wir kommen noch darauf zurück.

Der folgende Befehl im Listing gibt Ihnen vielleicht Rätsel auf. Nun, nicht immer ist es erwünscht, daß zu jeder Zeit ein Menüpunkt anwählbar ist. Ein anderer Fall ist der, daß der Anwender einen Modus über einen Eintrag einschalten kann (z.B. Fettschrift bei einer

Textverarbeitung) und das sichtbar gemacht werden soll. Im ersteren Fall bietet es sich an, den Eintrag hell darzustellen, im zweiten, vor dem Menüpunkt ein Häkchen (Check-Mark) zu setzen, das den eingeschalteten Modus signalisiert. Der folgende Befehl macht's möglich:

MENU Nr,Modus

Die Variable »Nr« enthält die Nummer des Eintrags im Array, der anders dargestellt werden soll. Das ist leider nicht sehr komfortabel, muß man doch bei der Programmerstellung berücksichtigen, welcher Menüeintrag an welcher Stelle im Stringarray zu finden ist. Programmierer, die die Menüleiste mit Hilfe eines *Resource-Construction-Sets* erstellen, haben es einfacher. Sie geben jedem Eintrag einfach einen Namen, durch den er später sehr leicht wieder identifiziert werden kann.

Als Modus sind untenstehende Werte möglich:

Modus	**bewirkt...**
0	das Löschen eines Check-Marks
1	das Setzen eines Check-Marks vor einen Menüpunkt
2	das Darstellen eines Menüpunktes in heller Schrift (nicht aktivierbar)
3	das Darstellen eines Menüpunktes in normaler Schrift (aktivierbar)

Die Programmzeile veranlaßt also die Darstellung des Menüeintrags, mit der Nummer 15 des Stringarrays, in heller Schrift.

```
  DEFFILL 1,3,6
  BOUNDARY 1                              !Umrahmung einschalten
  PBOX 20,0,620,300
  TEXT 30,33,150,"OUTBOX"
  '
  DEFFILL 1,3,3
  PBOX 120,50,520,250
  TEXT 125,80,"INBOX"
  '
  DEFTEXT 1,0,0,13                        !Normaler Textstil
  PRINT AT(3,20);"Scancode der Taste  :    "
  PRINT AT(3,21);"ASCII-Code der Taste:    "
  '
  PRINT AT(40,20);"X-Position des Mauscursors:     "
  PRINT AT(40,21);"Y-Position des Mauscursors:     "
RETURN
```

Der Rest des Unterprogramms baut die übrigen Teile des Bildschirms auf, wir können auf die Erklärung wohl verzichten.

Die folgende Prozedur wartet auf voreingestellte Ereignisse und veranlaßt bei ihrem Eintreten den Sprung in entsprechende Unterprogramme.

```
PROCEDURE warten
DO
  IF pfl&=1
    PRINT AT(68,20);USING "###",MOUSEX
    PRINT AT(68,21);USING "###",MOUSEY
  ENDIF
  ON MENU
LOOP
RETURN
```

Das Unterprogramm kann in den meisten Fällen aus nur drei Zeilen bestehen, es würde dann so aussehen:

```
DO
ON MENU
LOOP
```

Unser Programm zeigt Ihnen aber auf Wunsch auch die aktuelle Position des Mauscursors an. Darum ist noch die Abfrage des Flags »pfl&« erforderlich. Wie schon erwähnt, wartet das Programm auf vom Anwender vorgegebene Ereignisse. Was es damit auf sich hat, erfahren Sie im nächsten Abschnitt.

3.2 Ereignisse unter GEM

Ein Ereignis ist im menschlichen Leben immer etwas Besonderes. Wir verbinden bestimmte Ereignisse mit bestimmten Gefühlen, so ist für den einen eine Scheidung ein positives, für einen anderen ein negatives Ereignis. Ganz anders verhält es sich bei unserem ST. Er ist, wie alle Vertreter seiner Gattung, ein recht sturer Geselle und reagiert nur auf Ereignisse, die ihm das Programm vorgibt. Die Ereignisse, auf die er reagiert, haben wir im folgenden Teil einmal zusammengestellt. Da GFA-Basic 3.0 sowohl eigene als auch AES-nahe Befehle zur Ereignisverwaltung kennt, erfolgt die Erklärung im Zusammenhang.

Achtung: Der GFA-Befehl »ON MENU« wartet, im Gegensatz zu den AES-Funktionen, nicht auf das Eintreffen eines vorgegebenen Ereignisses. Es ist daher möglich, ihn auch in rechenintensiven Programmteilen aufzurufen, um evtl. auf Benutzeranforderungen zu reagieren. Die entsprechenden AES-Funktionen warten im Gegensatz dazu immer auf das bzw. die vorgegebenen Ereignisse. Tritt kein entsprechendes Ereignis ein bzw. läßt es sich nicht erzeugen, hilft nur noch die »Reset«-Taste.

Bei Verwendung der GFA-Befehle bedient sich unserer Meinung nach das Basic des EVNT_MULTI-Aufrufs unter Angabe einer recht kurzen Timerzeit. Für diese Vermutung spricht, daß es keinen GFA-Befehl für den AES-Timer gibt, sondern GFA sich bei der Timerprogrammierung des Interrupts bedient. Dazu später mehr.

Die untenstehende Tabelle gibt einen Überblick über die verfügbaren Möglichkeiten. Die Namen der GFA-Aufrufe wurden in Klammern gesetzt.

Ereignis	**tritt ein, wenn...**
EVNT_KEYBD (ON MENU KEY)	eine Taste betätigt wurde,
EVNT_BUTTON (ON MENU BUTTON)	ein oder mehrere Mausknöpfe gedrückt wurden,
EVNT_MOUSE (ON MENU IBOX & ON MENU OBOX)	der Mauszeiger einen vorher definierten Bildschirmbereich betritt oder verläßt,
EVNT_MESAG (ON MENU MESSAGE)	eine Nachricht für das laufende oder im Hintergrund wartende Programm übergeben werden soll,
EVNT_TIMER EVERY...GOSUB	wenn eine vorher bestimmte Zeit abgelaufen ist.

AES kann uns also, auf Verlangen, eine ganze Menge Arbeit abnehmen. Der Vorteil bei der Verwendung von GFA-Befehlen liegt darin, daß das Programm auch bei der Abarbeitung längerer Routinen ab und zu nachfragen kann, ob etwas eingetreten ist, das wesentlich wichtiger ist oder sogar die zur Zeit in Arbeit befindlichen Berechnungen überflüssig macht. Zugegeben, das läßt sich auch über einen entsprechenden AES-Aufruf (EVNT_MULTI) bewerkstelligen, nur ist er wesentlich umständlicher zu programmieren.

Bevor wir zur Erklärung der einzelnen AES- und GFA-Aufrufe kommen, einige Erläuterungen zu den GFA-Befehlen.

Der Befehl »ON MENU« reagiert nur auf vorher angegebene Ereignisse. Ein Ereignis, auf welches »ON MENU« reagieren soll, wird immer durch »ON MENU......GOSUB« definiert. Ausnahme: »ON MENU GOSUB ...« reagiert lediglich auf das Anwählen von Einträgen der Menüleiste. Bei allen Definitionen folgt nach »GOSUB« ein beliebiger Unterprogrammname. Ist kein Unterprogramm dieses Namens vorhanden, erfolgt auch keine Verzweigung. Das Basic sendet in diesem Fall keine Fehlermeldung. Ist, und das kommt vor, die Reaktion auf ein vorher definiertes Ereignis nicht mehr erwünscht, setzt man das Sprungziel durch erneuten Aufruf auf ein Unterprogramm, das nichts zu tun hat. Sie werden das im folgenden noch kennenlernen. Nach dem Eintreffen des Ereignisses

findet man in einem GFA-Array eine nähere Beschreibung. Das Array heißt »MENU( )«. Als Index sind die Zahlen von –2 bis 15 zugelassen. Bei der Programmierung der Drop-down-Menüs haben folgende Feldeinträge besondere Bedeutung:

Eintrag	Bedeutung
MENU(–1)	gibt die Adresse der Drop-down-Menü-Struktur im Speicher zurück.
MENU(0)	enthält nach der Wahl eines Eintrags den Index im Eintragsarray.
MENU(4)	enthält die Objektnummer des Menütitels.
MENU(5)	enthält die Objektnummer des angeklickten Menüeintrags.

Der Eintrag MENU(–2) gibt die Adresse des Messagebuffers, eine Einrichtung des AES, die wir bei der Fensterprogrammierung noch näher beschreiben werden, zurück. Es folgt die Programmierung der einzelnen Ereignisaufrufe unter AES und GFA. Parameter, die Sie übergeben müssen, haben den Kennbuchstaben P, Parameter, die Sie zurückerhalten, den Buchstaben R.

Der erste Aufruf bezieht sich auf die Tastatur. Er erfolgt unter AES durch folgenden Befehl:

R&=EVNT_KEYBD()

R& enthält den ASCII-Code der gedrückten Taste im Lowbyte und den Scan-Code im Highbyte des Wortes.

Der GFA-Aufruf sieht so aus:

ON MENU KEY GOSUB ...

In Menu(xx) sind folgende Angaben zu finden:

MENU(14) ist gleichzusetzen mit »R&« des AES-Aufrufes.
MENU(13) Status der Umschalttasten
1 rechte [Shift]-Taste gedrückt
2 linke [Shift]-Taste gedrückt
4 [Control]-Taste gedrückt
8 [Alternate]-Taste gedrückt

oder entsprechende Zwischenwerte.

R1=EVNT_BUTTON(P1,P2,P3,R2,R3,R4,R5)

P1	Anzahl der Mausklicks, auf die gewartet werden soll, z.B. Doppelklick
P2	Maske für die Mausknöpfe: 1 linker Mausknopf 2 rechter Mausknopf 3 beide Mausknöpfe
P3	Status des Mausknopfes bzw. der Mausknöpfe, die das Ereignis auslösen sollen: 0 Knopf nicht gedrückt 1 Knopf gedrückt

Nach Eintritt des Ereignisses erhalten Sie folgende Parameter zurück:

R1	Anzahl der Mausklicks
R2	X-Koordinate des Mauspfeils beim Eintreten des Ereignisses
R3	Y-Koordinate des Mauspfeils beim Eintreten des Ereignisses
R4	Betätigte Maustaste(n)
R5	Status der Umschalttasten: 1 rechte [Shift]-Taste gedrückt 2 linke [Shift]-Taste gedrückt 4 [Control]-Taste gedrückt 8 [Alternate]-Taste gedrückt

oder entsprechende Zwischenwerte.

Achtung: Sie müssen, wie oben ersichtlich, die *Anzahl* der Mausklicks übergeben. Haben Sie als Anzahl zwei angegeben, meldet sich das AES auch nach einem Mausklick beim Programm zurück. Es wartet nämlich nur einen bestimmten Zeitraum auf den zweiten Klick. Der Zeitraum kann über das AES oder über das Kontrollfeld bestimmt werden. Ein Programm muß daher, wenn sowohl ein als auch zwei Klicks möglich sein sollen, immer die Anzahl der Klicks auswerten.

Unter GFA stellt sich das Ganze so dar:

ON MENU BUTTON P1,P2,P3 GOSUB...

Die Parameter »P1« bis »P3« haben die gleiche Bedeutung wie beim AES-Aufruf. Die Rückgabeparameter erhält man, nach dem Eintritt des Ereignisses, wieder im MENU-Array.

MENU(15)	Anzahl der Mausklicks
MENU(10)	X-Koordinate des Mauspfeils beim Eintreten des Ereignisses
MENU(11)	Y-Koordinate des Mauspfeils beim Eintreten des Ereignisses
MENU(12)	Betätigte Maustaste(n); linke = 1, rechte = 2, beide = 3

MENU(13) Status der Umschalttasten:
1 rechte [Shift]-Taste gedrückt
2 linke [Shift]-Taste gedrückt
4 [Control]-Taste gedrückt
8 [Alternate]-Taste gedrückt

oder entsprechende Zwischenwerte.

Das nächste Ereignis tritt ein, wenn der Mauscursor ein bestimmtes Rechteck betritt oder verläßt. Zunächst der AES-Aufruf:

~EVNT_MOUSE(P1,P2,P3,P4,P5,R1,R2,R3,R4)

P1 Entscheidet, ob das Ereignis beim Betreten oder Verlassen des definierten Rechtecks auftreten soll
0 signalisiert das Betreten
1 signalisiert das Verlassen
P2 X-Koordinate der linken oberen Ecke des Rechtecks
P3 Y-Koordinate der linken oberen Ecke des Rechtecks
P4 Breite des Rechtecks
P5 Höhe des Rechtecks

Sie erhalten folgende Daten nach dem Ereignis zurück:

R1 X-Koordinate des Mauspfeils beim Eintreten des Ereignisses
R2 Y-Koordinate des Mauspfeils beim Eintreten des Ereignisses
R3 Betätigte Maustaste(n); 1 = rechte, 2 = rechte, 3 = beide
R4 Status der Umschalttasten:
1 rechte [Shift]-Taste gedrückt
2 linke [Shift]-Taste gedrückt
4 [Control]-Taste gedrückt
8 [Alternate]-Taste gedrückt

oder entsprechende Zwischenwerte.

Der Aufruf mit GFA-Befehlen heißt entweder:

ON MENU INBOX P1,P2,P3,P4,P5 GOSUB ...

oder

ON MENU OBOX P1,P2,P3,P4,P5 GOSUB ...

Es können maximal zwei unabhängige Rechtecke definiert werden. Vorstellbar wäre, eines als »OBOX« und eines als »IBOX« zu definieren, möglich ist aber auch die Definition von zweimal »OBOX« oder »IBOX«. Die Variable »P1« enthält die Nummer des zu definierenden Rechtecks. Die Parameter »P2« bis »P5« haben die gleiche Bedeutung wie beim AES-Aufruf. Das Programm kann folgende Rückgabewerte aus dem MENU-Array abfordern:

MENU(10) X-Koordinate des Mauspfeils beim Eintreten des Ereignisses

MENU(11) Y-Koordinate des Mauspfeils beim Eintreten des Ereignisses

MENU(12) Betätigte Maustaste(n); 1 = rechte, 2 = rechte, 3 = beide

MENU(13) Status der Umschalttasten:
1 rechte `Shift`-Taste gedrückt
2 linke `Shift`-Taste gedrückt
4 `Control`-Taste gedrückt
8 `Alternate`-Taste gedrückt

oder entsprechende Zwischenwerte.

Über den nächsten Aufruf läßt sich nach dem Ablauf einer definierbaren Zeit ein Unterprogramm abarbeiten.

Der entsprechende AES-Aufruf läßt sich unter GFA wie folgt bewerkstelligen:

~EVNT_TIMER(P1)

P1 Anzahl der Millisekunden, die gewartet werden soll.

AES meldet sich nach der in »P1« angegebenen Zeitspanne zurück. Das GFA-Basic kennt aber noch einen einfacher zu handhabenden Befehl. Er ist wesentlich flexibler als der AES-Aufruf. Er heißt:

EVERY Anzahl **GOSUB ...**

In der Variablen »Anzahl« wird die Zeit (in 200stel-Sekunden-Schritten) angegeben, die immer bis zum Unterprogrammaufruf verstreichen soll. Bedenken Sie bitte folgendes:

Je länger der Ablauf der angesprungenen Routine dauert, um so länger sollte die eingestellte Zeit sein. Das Unterprogramm wird sonst aus sich selbst heraus aufgerufen, und irgendwann kommt die Meldung: »Speicher voll«. Sie können das umgehen, indem Sie bei Aufruf eines Unterprogramms, das bei der Arbeit nicht gestört werden soll, den Befehl »EVERY STOP« einsetzen. GFA stellt nach diesem Aufruf die Abfrage des Systemtimers vorerst ein. »Vorerst« deshalb, weil der Befehl »EVERY CONT« die Abfrage mit dem »alten« Wert erneut aufnimmt. Eine kleine Bemerkung für die Profis unter Ihnen: GFA wartet nicht, sondern läßt sich vom Systemtimer unterbrechen.

Hinweis: Gestalten Sie die Routine, die vom Timer aufgerufen wird, nicht zu umfangreich, die Geschwindigkeitseinbußen des Hauptprogramms sind sonst sehr erheblich.

Es gibt noch eine weitere Variante der Zeitsteuerung:

Der AES-Aufruf wartet bekanntlich bis zur abgelaufenen Zeit und gibt erst danach die Kontrolle wieder an das Programm zurück. Derjenige unter Ihnen, der den AES-Aufruf weitestgehend mit einem GFA-Aufruf nachbilden möchte, kann die folgende Möglichkeit verwenden:

AFTER Anzahl **GOSUB ...**

Nach Ablauf der in »Anzahl« festgelegten Zeit verzweigt das Programm zur angegebenen Routine. Die Zeitmessung läßt sich durch den Befehl »AFTER STOP« unterbrechen und mit »AFTER CONT« wieder fortsetzen.

Ein weiterer GFA-Befehl bzw. GFA-AES-Aufruf bezieht sich auf Nachrichten, die das Betriebssystem dem laufenden Programm zukommen läßt. Was Nachrichten sind und wie ein Programm auf sie zu reagieren hat, besprechen wir genauer im Kapitel über die Fensterverwaltung.

~**EVNT_MESAG**(adr_buffer%)

In der Variablen »adr_buf%« übergibt man der Funktion die Adresse eines 16 Byte großen Puffers, das entspricht der Länge von acht Wörtern. In ihm legt das AES nach dem Eintreffen einer Nachricht genauere Informationen ab. Im ersten Wort des Puffers steht immer eine Nummer, die die eingetroffene Nachricht eindeutig kennzeichnet. Das Wählen eines Eintrags des Drop-down-Menüs wird zum Beispiel immer durch den Wert 10 im ersten Eintrag des Message-Buffers signalisiert. Das vierte Wort des Puffers enthält die Objektnummer des Menütitels, das fünfte die Objektnummer des angewählten Menüpunktes. Die Menüleiste fragt man also bei der Verwendung von AES-Funktionen über die obenstehende Funktion ab.

Den Anwendern, die die GFA-Befehle nutzen möchten, bietet sich folgende Definition an:

ON MESSAGE GOSUB....

Auch bei diesem Aufruf wird ein *Message-Buffer* beim Eintreffen einer Nachricht belegt. Er befindet sich im »Menu«-Array und besitzt die Indizes 1 bis 8. Die genaue Belegung finden Sie ebenfalls im Kapitel über die Fensterprogrammierung.

Sehr oft ist es wünschenswert, daß ein Programm auf mehrere Ereignisse wartet und immer auf das zuerst eintreffende reagiert. Hält man sich an die reinen GFA-Befehle, stellt sich das Ganze recht einfach dar. Man ruft einfach die vorgestellten Befehle nacheinander auf und wartet in einer Schleife auf die Dinge, die das AES mitzuteilen hat. Ein Beispiel:

```
ON MENU GOSUB Menueleiste
ON MENU MESSAGE GOSUB Fenster
EVERY 100 GOSUB Uhr
'
DO
ON MENU
LOOP
'
' Ab Hier folgen die Unterprogramme
```

In der Schleife wartet das Programm auf Anklicken eines Eintrags der Menüleiste, das Eintreffen einer Nachricht und das Verstreichen einer bestimmten Zeit. Wer mit den GFA-AES-Funktionen arbeitet, hat es bei der Mehrfachabfrage etwas schwerer. Die entsprechende Funktion ist nämlich recht komplex. Sie heißt »EVNT_MULTI«.

R1=EVNT_MULTI(P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15, P16,R2,R3,R4,R5,R6,R7)

P1 — Es handelt sich um ein Flag-Register. In ihm wird über gesetzte Bits bestimmt, auf welche Ereignisse gewartet werden soll. Die Zuordnung der Bits ist folgende:
Bit 0 Tastaturereignis
Bit 1 Mausknopfereignis
Bit 2 Mausereignis, erstes Rechteck
Bit 3 Mausereignis, zweites Rechteck
Bit 4 Nachricht
Bit 5 Timerereignis

P2 — Anzahl der Mausklicks, auf die gewartet werden soll, z.B. Doppelklick

P3 — Maske für die Mausknöpfe:
1 linker Mausknopf
2 rechter Mausknopf
3 beide Mausknöpfe

P4 — Status des Mausknopfes bzw. der Mausknöpfe, die das Ereignis auslösen sollen:
0 Knopf nicht gedrückt
1 Knopf gedrückt

P5 — Entscheidet, ob das Ereignis beim Betreten oder Verlassen des ersten definierten Rechtecks auftreten soll:
2 signalisiert das Betreten
1 signalisiert das Verlassen

P6 — X-Koordinate der linken oberen Ecke des ersten Rechtecks

P7 — Y-Koordinate der linken oberen Ecke des ersten Rechtecks

P8 — Breite des ersten Rechtecks

P9 — Höhe des ersten Rechtecks

P10 bis P14 — wie oben, für das zweite mögliche Rechteck

P15 — Adresse des Message-Buffers

P16 — Anzahl der Millisekunden, die gewartet werden soll.

Als Rückgabe erhalten Sie:

R1 — Es handelt sich wie bei P1 um ein Flag-Register. Die Belegung ist die gleiche, nur gibt dieses Register das eingetretene Ereignis zurück.

R2 — X-Koordinate des Mauspfeils beim Eintritt des Ereignisses

R3 — Y-Koordinate des Mauspfeils beim Eintritt des Ereignisses

R4 — Betätigte Maustaste(n)

R5 — Status der Umschalttasten:
1 rechte [Shift]-Taste gedrückt
2 linke [Shift]-Taste gedrückt
4 [Control]-Taste gedrückt
8 [Alternate]-Taste gedrückt

oder entsprechende Zwischenwerte.

R6 — Tastaturcode

R7 — Anzahl der Maustastenklicks

Wer sich diesen Aufruf etwas näher zu Gemüte führt, erkennt sehr leicht, daß es sich bei ihm um eine echte Zusammenfassung aller Events handelt. Daher gilt selbstverständlich alles bereits Gesagte (Geschriebene). Soviel zu den Ereignissen. Im Laufe des Programms werden Sie fast alle in Aktion antreffen. Machen wir daher mit dem Listing weiter.

Es folgt das Unterprogramm, das nach dem Anklicken eines Menüeintrags aufgerufen wird.

```
PROCEDURE menue_auswertung
  e&=MENU(0)                        !Array-Index
  e$=eintrag$(e&)
```

Die Indexnummer des Eintrags im Feld steht, wie Sie schon wissen, nach dem Anklicken immer in MENU(0). Das Programm übernimmt ihn in die Variable »e&«, wir waren einfach zu bequem, immer MENU(0) zu tippen. Der entsprechende Feldeintrag wird danach in die Stringvariable »e$« kopiert.

```
IF e$=" ÜBER DAS PROGRAMM "
 s$="    MENÜLEISTEN DEMO|          VON| |"
 ALERT 1,s$+"JENS MUUS & WILHELM BESENTHAL",1," OK ",b&
```

Das Programm vergleicht jetzt den Inhalt von »e$« mit den Original-Strings und leitet dann entsprechende Aktionen ein. Die Copyrightmeldung realisiert das Programm über eine Alarmbox. Ihr Aufbau ist wie folgt:

ALERT n,Überschrift,n1,Buttontext,Variable

n	Zahl von 0 bis 3, 0 = kein Zeichen, 1 = !, 2 = ?, 3 = STOP
Überschrift:	4 Zeilen mit maximal je 30 Zeichen
n1	Gibt an, welcher Button stärker umrandet sein soll
Buttontext	Text der einzelnen Buttons, maximal je 8 Zeichen lang

Die Größe der Box ist wie der Ausschnitt des Bildschirms, in dem die Box erscheint, fest vorgegeben. Damit Sie die Textzeilen selbst gestalten können, muß ein »|« (senkrechter Strich) als Steuerzeichen verwendet werden. Dieses Zeichen markiert das Ende der Textzeile. Der nachfolgende Text wird dann bis zum nächsten Steuerzeichen in der zweiten Zeile dargestellt. Beachten Sie bitte die Höchstanzahl von vier Textzeilen pro Box. Soll eine Zeile übersprungen werden, so muß dem ersten »|« ein Leerzeichen folgen, das nächste Zeichen ist dann wieder »|«.

Das Steuerzeichen hat aber noch eine andere Aufgabe. Es trennt nämlich die einzelnen Buttontexte voneinander. Jeder Buttontext darf eine maximale Länge von acht Zeichen haben. Der Parameter »n1« gibt an, welcher Button stärker umrandet sein soll. Dieser Button kann dann auch über die [Return]-Taste aktiviert werden. Hat »n1« den Wert Null, so weisen alle Buttons einen gleich starken Rand auf.

```
  '
  ' INBOX ZULASSEN ODER SPERREN
  '
ELSE IF e$=" INBOX  AUS "
  eintrag$(e&)=" INBOX  EIN "
  ~MENU_TEXT(MENU(-1),MENU(5),eintrag$(e&)+CHR$(0))
  ON MENU IBOX 1,120,50+19,400,200 GOSUB inbox
```

Wählt der Benutzer des Programms den Eintrag »INBOX AUS«, ändert das Programm den Namen des Eintrags in »INBOX EIN« um. Die Namensänderung erfolgt mit Hilfe einer AES-Funktion. Sie heißt »MENU_TEXT« und soll etwas näher betrachtet werden.

~MENU_TEXT(Adr%,Nr,Neuer_N$)

Adr%	Enthält die Adresse der Menüstruktur
Nr	Objektnummer des Objekts, dessen Name geändert werden soll
Neuer_N$	Enthält den neuen Namen; der String muß mit ASCII null enden

Achtung: Der neue Name darf nie länger als der ursprüngliche Name sein. Es kann sonst passieren, daß das Programm oder sogar der Rechner die Zusammenarbeit mit Ihnen einstellt.

Wichtig ist auch noch, zu beachten, daß es sich um die *Objektnummer* und nicht um das *Indiz* des Stringfeldes handelt, das übergeben wird. Wer damit noch nicht allzuviel anzufangen weiß, der lese später den zweiten Teil dieses Kapitels.

Die INBOX soll also eingeschaltet werden, das macht die nächste Zeile der Auswertung. Sie legt die Ausmaße der Box und das Sprungziel, das bei Betreten der Box durch den Mauszeiger aufgerufen werden soll, fest.

Die nächsten Zeilen schalten die Reaktion auf ein Inbox-Ereignis, nach nochmaligem Anklicken des Eintrags, wieder aus.

```
ELSE IF e$="  INBOX  EIN "
  eintrag$(e&)="  INBOX  AUS "
  ~MENU_TEXT(MENU(-1),MENU(5),eintrag$(e&)+CHR$(0))
  ON MENU IBOX 1,120,50+19,400,200 GOSUB nichts
```

Nach dem Ändern des Eintrags in »INBOX AUS« ruft das Programm den INBOX-Befehl mit den gleichen Koordinaten wie bei »INBOX EIN« erneut auf. Als Sprungziel übergibt es eine Prozedur namens »nichts«. Und genau das passiert in ihr, nämlich absolut nichts. Die Prozedur muß wirklich existieren, sonst wird das alte Sprungziel nicht gelöscht, die Inbox bleibt weiterhin aktiv.

```
  '
  ' OUTBOX ZULASSEN ODER SPERREN
  '
ELSE IF e$="  OUTBOX AUS "
  eintrag$(e&)="  OUTBOX EIN "
  ~MENU_TEXT(MENU(-1),MENU(5),eintrag$(e&)+CHR$(0))
  ON MENU OBOX 2,20,0+19,600,300 GOSUB outbox
ELSE IF e$="  OUTBOX EIN "
  eintrag$(e&)="  OUTBOX AUS "
  ~MENU_TEXT(MENU(-1),MENU(5),eintrag$(e&)+CHR$(0))
  ON MENU OBOX 2,20,0+19,600,300 GOSUB nichts
```

Die obenstehenden Zeilen bilden das Gegenstück zu INBOX, sie schalten das Outbox-Ereignis ein oder aus. Wer die Erklärungen bei der Inbox verstanden hat, dem fällt es sicher nicht schwer, auch diese Zeilen schnell zu durchschauen.

```
ELSE IF e$="  TIMER EIN "
  MENU 15,3                          !>Timer ein< Nicht selektierbar
  MENU 14,2                          !>Timer aus< selektierbar
  EVERY 200 GOSUB uhr                !Jede Sekunde Uhr stellen
ELSE IF e$="  TIMER AUS "
  MENU 15,2                          !>Timer aus< Nicht selektierbar
  MENU 14,3                          !>Timer ein< selektierbar
  EVERY 1000 GOSUB nichts            !Uhraufruf-Stop
```

Durch die REM-Zeilen erklärt sich der Programmabschnitt wohl von selbst. Je nach gewünschtem Zustand des Timers werden die entsprechenden Einträge hell oder normal dargestellt.

```
ELSE IF e$="  POSITION "
  pfl&=pfl& XOR 1
  IF pfl&=1
    MENU e&,1
  ELSE
    MENU e&,0
  ENDIF
```

Auf Wunsch zeigt Ihnen das Programm die Mausposition an, das ist vor allem im Hinblick auf die möglichen negativen Koordinaten recht interessant. Werden die Koordinaten angezeigt, erscheint vor dem Menüeintrag ein *Check-Mark*. Soll der Anzeigemodus ausgeschaltet werden, reicht das nochmalige Wählen des Eintrags, um das Positionsflag »pfl&« zu löschen und das Check-Mark aus dem Eintrag wieder zu entfernen.

```
ELSE IF e$="  TASTATUR "
  tfl&=tfl& XOR 1
  IF tfl&=1
    MENU e&,1
    ON MENU KEY GOSUB tastatur
  ELSE
    MENU e&,0
    ON MENU KEY GOSUB nichts
  ENDIF
```

Ähnlich verhält es sich bei der Anzeige des »*Scan-Codes*« und des ASCII-Codes beim Drücken einer Taste der Tastatur. Entweder verzweigt das Programm in die Prozedur »tastatur«, in der die Ausgabe vorgenommen wird, oder in das Unterprogramm »nichts«, in dem nichts passiert.

```
ELSE IF e$="  ENDE "
  MENU KILL
  CLOSEW 0
  END
ENDIF
```

Bei Wahl des Menüpunkts ENDE schaltet das Programm die Menüabfrage ab und schließt Fenster null. Der Befehl »END« beendet das Programm.

MENU KILL

Soll die Abfrage der Menüleiste für immer eingestellt werden, so bedient man sich zweckmäßigerweise dieses Befehls. Die Abfrage läßt sich dann aber nie wieder während des Programmlaufs einschalten. Vorübergehendes Einstellen der Abfrage nimmt man am besten so vor:

ON MENU GOSUB nichts

Kommen wir zur letzten relevanten Zeile des Unterprogramms.

```
MENU OFF
RETURN
```

Wurde ein Eintrag eines Menüs gewählt, bleibt der Titel des Menüs nach wie vor invers. Diese Darstellungsart sollte bis zum Ende der Abarbeitung der entsprechenden Reaktion beibehalten werden. Sie signalisiert dem Benutzer, daß das Programm noch nicht wieder in der Lage ist, auf neue Anforderungen zu reagieren. Die inverse Darstellung läßt sich durch den untenstehenden Befehl ausschalten.

MENU OFF

Er wird, wie schon erwähnt, am besten erst dann aufgerufen, wenn alle Reaktionen abgeschlossen sind und das Programm zu neuen Schandtaten bereit ist.

Die Verzweigungen zu den jetzt folgenden Unterprogrammen, werden immer durch das Eintreten eines entsprechenden Ereignisses während eines ON MENU-Aufrufs ausgelöst.

```
PROCEDURE inbox
  ALERT 3," DER MAUSCURSOR IST | IN DER| | INBOX !",1," OK ",bu&
RETURN
PROCEDURE outbox
  ALERT 3," DER MAUSCURSOR IST | AUßERHALB DER| | OUTBOX !",1,
        " OK ",bu&
RETURN
```

Inbox- und Outbox-Ereignisse teilt Ihnen das Programm durch eine entsprechende Alert-Box mit.

```
PROCEDURE uhr
IF TIME$=ti$ AND VAL(RIGHT$(ti$,1))+1<10
  MID$(ti$,8)=RIGHT$(STR$(VAL(RIGHT$(ti$,1))+1),1)
ELSE
  ti$=TIME$
ENDIF
PRINT AT(24,23);"UHRZEIT:    ";ti$;
RETURN
```

Wir benutzen für die Zeitanzeige die Systemzeit, die in der Systemvariablen »TIME$« zu finden ist. Da die Systemzeit aber nur alle zwei Sekunden vom Rechner auf den neuesten Stand gebracht wird, wir aber die Uhrzeit alle Sekunde aktualisieren wollten, war noch etwas Aufwand erforderlich. Der Aufruf erfolgt in der Sekunde einmal. Die Programmzeilen überprüfen nun, ob die Zeit schon vom System angepaßt wurde, wenn nicht, nimmt das Programm das Stellen der Uhr auf dem Bildschirm selbst in die Hand.

```
PROCEDURE tastatur
  EVERY STOP                              !Uhr ausschalten
  scan_code&=MENU(14) DIV 256
  ascii_code&=MOD(MENU(14),256)
  PRINT AT(3,20);"Scancode der Taste  : ";
  PRINT USING "###",scan_code&
  PRINT AT(3,21);"ASCII-Code der Taste: ";
  PRINT USING "###",ascii_code&
  EVERY  CONT                             !Uhr wieder einschalten
RETURN
```

Die obenstehende Prozedur springt das Programm beim Eintreten eines Tastaturereignisses an. Da die Anzeige der zurückerhaltenen Werte, wie auch der Uhrzeit, über PRINT-Befehle erfolgt, verhindert der Befehl »EVERY STOP« den Aufruf der Uhr während der Abarbeitung dieses Unterprogramms. Fehlt der Befehl, kann es passieren, daß ein PRINT-Befehl nicht an der richtigen Stelle ausgeführt wird. Das Programm schaltet den Timer-Interrupt am Ende des Unterprogramms erneut ein.

```
PROCEDURE nichts
RETURN
```

Das ist die kürzeste Routine des Programms. Wir hoffen, daß Ihnen schon der Name mehr als »nichts« sagt. Sie sollten sonst vielleicht das Kapitel noch einmal durchlesen. Wir sind mit den Themen zu den Drop-down-Menüs fertig. Im nächsten Abschnitt geht es um Dialogboxen.

3.3 Dialogboxen

Ein weiteres wichtiges Element zur Programmgestaltung auf dem Atari ST stellen die Dialogboxen, auch Formulare genannt, dar. Ihre Verwaltung ist nicht immer ganz einfach, können sie doch einen recht komplexen Aufbau vom Programmierer erhalten. Einen vielleicht etwas aufgeblähten Vertreter seiner Art finden Sie zum Beispiel im Programm zur Dateiverwaltung. Für dieses Kapitel haben wir eine Dialogbox nebst Programm erstellt, die Ihnen das Warmwerden mit dem Thema nicht ganz so schwer macht. Bevor es losgeht, noch ein Hinweis:

Dialogboxen entwickelt man zweckmäßigerweise mit einem *Resource-Construction-Set*. Auf der Originaldiskette des GFA-Basic befindet sich ein solches Programm. Wir

beschreiben es im letzten Kapitel des Buches, da sonst die Zusammenhänge dieses Kapitels zu sehr getrennt würden.

Vor der Programmbeschreibung kommen erstmal wieder Grundlagen auf Sie zu. Alles können wir hier nicht beschreiben, sonst hätten Sie ein komplettes GEM-Buch vor sich liegen.

3.3.1 Objektstrukturen

Dialogboxen und auch Drop-down-Menüs bestehen aus einer Unzahl von Daten. Das muß so sein, denn der Programmierer erwartet ja vom AES, daß es seine einmal entwickelte Dialogbox auch genauso auf den Bildschirm bringt, wie er sie erstellte. Ein *Resource-Construction-Set* wandelt darum die Grafiken, die der Benutzer erzeugt, in Daten um, die das AES versteht. Aber nicht nur das. AES kennt verschiedene Hierarchien, die der Benutzer selbst festlegen darf. Dadurch ist es zum Beispiel möglich, daß man dem AES mitteilen kann, nur bestimmte Teile einer Dialogbox zu zeichnen. Die Struktur, die das AES kennt, nennt man auch Baumstruktur.

Jeder Baum beginnt bei den Wurzeln. Auf das AES bezogen spricht man vom *Wurzelobjekt*. Diesem Objekt sind alle anderen Objekte untergeordnet. Ein Objekt ist immer nur ein Teil einer Dialogbox oder eines Drop-down-Menüs, d.h., jeder String oder jeder Button stellt ein eigenes Objekt dar. Jedes Objekt darf *Unterobjekte* besitzen. Auf unseren Baum bezogen heißt das, daß von jedem Zweig andere Zweige wachsen können. Sägt man einen dicken Ast ab, fallen auch die dünneren Äste und ebenfalls deren Äste mit herunter. Genauso beim AES. Der Programmierer kann z.B. bestimmen, daß Teile seines *Menübaumes* hinzukommen oder entfernt werden. Das AES kann auch mehrere Menübäume verwalten, im Programm über die Diskettenbefehle haben wir davon Gebrauch gemacht. Das oben Gesagte werden Sie um so leichter verstehen, wenn wir Ihnen einmal die Struktur von Objektbäumen zeigen.

Die Strukturen stehen sequentiell, d.h. hintereinander, im Speicher. Damit das AES nun feststellen kann, welches Objekt zu welchem Zweig gehört, ist jedes Objekt in einem 24 Byte großen Feld beschrieben. Einige Objekte bedürfen noch weiterer Daten, es sind Zeiger auf andere Adressen, die das Objekt näher beschreiben. Die Einträge haben immer Wortlänge.

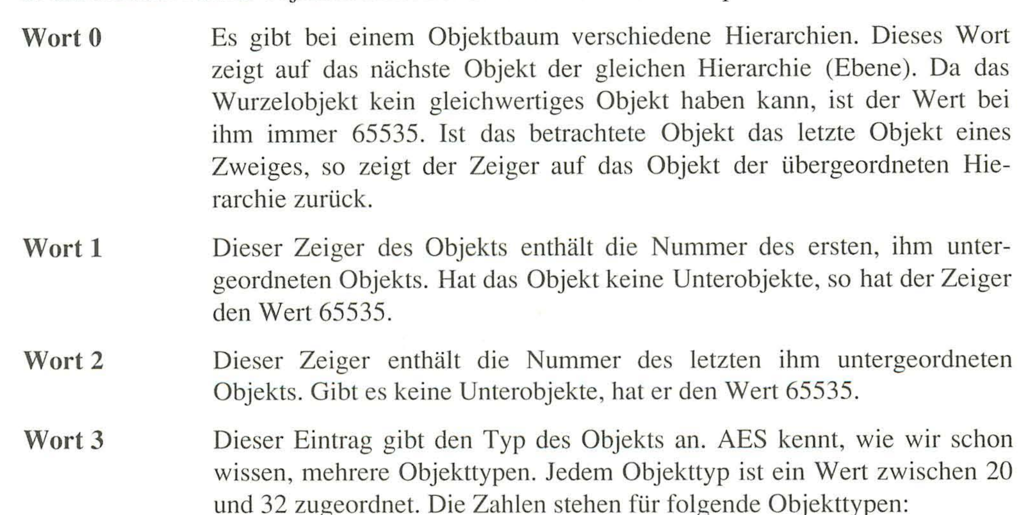
Die Belegung des Feldes und die Namen der Einträge:

Wort	Name	Inhalt
0	OB_NEXT	Nächstes Objekt dieses Zweiges.
1	OB_HEAD	Nummer des ersten, diesem Objekt untergeordneten Objekts.
2	OB_TAIL	Nummer des letzten, diesem Objekt untergeordneten Objekts.

Wort	Name	Inhalt
3	OB_TYPE	Objekttyp, Zahl zwischen 20 und 32.
4	OB_FLAGS	Flagregister dieses Objekts.
5	OB_STATE	Statusregister dieses Objekts.
6	OB_SPEC	Je nach Objekt Teil eines Zeigers oder weitere Information, wird unten näher beschrieben.
7		Siehe Wort 6.
8	OB_X	X-Koordinate, relativ zum übergeordneten Objekt.
9	OB_Y	Y-Koordinate, relativ zum übergeordneten Objekt.
10	OB_W	Breite des Objekts.
11	OB_H	Höhe des Objekts.

GFA 3.0 kennt Funktionen, die den gleichen Namen wie die Einträge tragen. Sie werden ihren sinnvollen Einsatz weiter unten, bei der Programmbeschreibung, kennenlernen.

Aus der Beschreibung der Wörter 0 bis 2 ist zu ersehen, daß jedes Objekt eine Nummer hat. Sie wird beim Erstellen mit dem Construction Set von diesem vergeben. Betrachten wir die Einträge des Parameterblocks noch etwas genauer. Beachten Sie dabei bitte, daß die Zeiger in den meisten Fällen Objektnummern und keine Positionen im Speicher enthalten.



Wort 0 Es gibt bei einem Objektbaum verschiedene Hierarchien. Dieses Wort zeigt auf das nächste Objekt der gleichen Hierarchie (Ebene). Da das Wurzelobjekt kein gleichwertiges Objekt haben kann, ist der Wert bei ihm immer 65535. Ist das betrachtete Objekt das letzte Objekt eines Zweiges, so zeigt der Zeiger auf das Objekt der übergeordneten Hierarchie zurück.

Wort 1 Dieser Zeiger des Objekts enthält die Nummer des ersten, ihm untergeordneten Objekts. Hat das Objekt keine Unterobjekte, so hat der Zeiger den Wert 65535.

Wort 2 Dieser Zeiger enthält die Nummer des letzten ihm untergeordneten Objekts. Gibt es keine Unterobjekte, hat er den Wert 65535.

Wort 3 Dieser Eintrag gibt den Typ des Objekts an. AES kennt, wie wir schon wissen, mehrere Objekttypen. Jedem Objekttyp ist ein Wert zwischen 20 und 32 zugeordnet. Die Zahlen stehen für folgende Objekttypen:

Wert	Objekttyp
20	BOX
21	TEXT
22	BOXTEXT
23	IMAGE
24	PROGDEF
25	IBOX
26	BUTTON
27	BOXCHAR
28	STRING
29	FTEXT
30	FBOXTEXT
31	ICON
32	TITEL

Aus Platzgründen können wir an dieser Stelle nur die wichtigsten, immer wiederkehrenden Objekttypen kurz beschreiben. Möchten Sie mehr darüber erfahren, lesen Sie bitte in entsprechender GEM-Literatur nach.

Einige Objekttypen benötigen noch weitere Parameter. Es existiert dann ein Zeiger, gebildet aus den Wörtern 6 und 7, auf einen weiteren Parameterblock. Er beschreibt den Objekttyp näher. Tritt dieser Fall ein, so weisen wir bei der Objektbeschreibung darauf hin.

BOX Dieser Objekttyp stellt, wie der Name schon sagt, einen Kasten dar. Er wird in wohl allen Objektstrukturen auftauchen. Wort 6 enthält einen Wert, der die Dicke des Randes beschreibt. Es bedeutet: 1–128 Randdicke nach innen, 129–255 Randdicke nach außen. Wort 7 enthält die Farbe des Objekts. Da es sich auch hierbei um einen Wert mit Wortlänge handelt, stehen insgesamt 16 Bit zur Verfügung.

Die einzelnen Bits haben folgende Bedeutung:

Bit 0–3	Stellt die Füllfarbe dar.
Bit 4–6	Ergibt den Füllmodus, je höher der Wert, um so deckender wird gefüllt
Bit 7	Repräsentiert den Schreibmodus 0 = durchsichtig 1 = deckend
Bit 8–11	Textfarbe
Bit 12–15	Farbe des Randes

Folgende Farben sind möglich:

Farbcode	Farbe
0	Weiß
1	Schwarz
2	Rot
3	Grün
4	Blau
5	Cyan
6	Gelb
7	Magenta
8	Weiß
9	Schwarz
10	Hellrot
11	Hellgrün
12	Hellblau
13	Hellcyan
14	Hellgelb
15	Hellmagenta

Es gibt also jede Menge Möglichkeiten bei der Gestaltung einer Box. Weiter geht's mit der Beschreibung weiterer Objekttypen.

TEXT Das Objekt besteht aus Grafiktext. Wort 6 und Wort 7 enthalten eine Adresse, ab der die TEDINFO-Struktur (nähere Beschreibung des Textes) im Speicher zu finden ist.

BOXTEXT Dies ist ein Kasten, der einen Text enthält. Wort 6 und Wort 7 zeigen auf eine TEDINFO-Struktur.

IMAGE Dieses Objekt ist eine Grafik. Wort 6 und Wort 7 zeigen auf eine BITBLK-Struktur (nähere Beschreibung der Grafik).

IBOX Ein Rechteck, dessen Inhalt unsichtbar ist. Durch entsprechende Einstellung der Randdicke kann der Rand entweder sichtbar oder ebenfalls unsichtbar sein. Dieser Typ wird oft als Wurzelobjekt verwendet.

BUTTON Ein Text wird in einem Rechteck zentriert ausgegeben. Wort 6 und 7 zeigen auf den Text, der mit einem Nullbyte enden muß.

BOXCHAR In einem Rechteck befindet sich, zentriert, ein einziger Buchstabe. Der Buchstabe steht als ASCII-Code in Wort 6, Wort 7 enthält die Objektfarbe.

STRING Dieses Objekt enthält Grafiktext. Wort 6 und Wort 7 zeigen auf den auszugebenden Text, der mit einem Nullbyte abgeschlossen werden muß.

FTEXT Es handelt sich um formatierten Grafiktext. Wort 6 und Wort 7 bilden einen Zeiger auf eine TEDINFO-Struktur.

FBOXTEXT In einem Rechteck eingeschlossener formatierter Grafiktext. Wort 6 und Wort 7 zeigen auf eine TEDINFO-Struktur.

ICON Mit diesem Objekttyp lassen sich Icons darstellen. Wort 6 und Wort 7 zeigen auf eine ICONBLK-Struktur.

TITEL Es handelt sich um Grafiktext. Wort 6 und Wort 7 zeigen auf den Text, der mit einem Nullbyte abgeschlossen sein muß.

Soweit zu den Objekttypen. Nächste Station unseres Schnelldurchlaufs ist das Wort 4, das Flagregister.

Wort 4 Damit läßt sich die Bestimmung eines Objekts festlegen. Ein Beispiel wäre die Editiermöglichkeit eines vorgegebenen Textes. Das Flagregister ist bitorientiert, d.h., jedes Bit steht für eine Eigenart des Objekts. Es dürfen auch mehrere Bits gesetzt sein, bedenken Sie aber, daß einige Kombinationen keinen Sinn machen. Es werden nicht alle Bits dieses Wortes benötigt, wir beschreiben auch nur die immer wiederkehrenden.

Bit	Dez. Wert	Bedeutung
0	1	SELECTABLE: Es handelt sich um ein Eingabeobjekt. Wird es angewählt, erfolgt die reverse Darstellung.
1	2	DEFAULT: Dieses Objekt kann auch mit der [RETURN]-Taste angewählt werden. Es darf immer nur *ein* Objekt eines Objektbaumes so gekennzeichnet sein.
2	4	EXIT: Wird dieses Objekt ausgewählt und befindet sich der Mauscursor nach dem Loslassen der linken Maustaste noch über dem Objekt, geht die Kontrolle wieder an das Programm zurück. Damit das Anklicken Wirkung zeigt, muß auch das Bit 0 gesetzt sein.
3	8	EDITABLE: Dieses Objekt ist vom Bediener änderbar, d.h., der Cursor läßt sich darin plazieren, zum Beispiel bei TEXT.

Bit	Dez. Wert	Bedeutung
4	16	RBUTTON: Dieses Flag hat nur bei einer Gruppe von Objekten (es müssen mindestens zwei sein) Bedeutung. Ein Knopf, der vorher revers dargestellt wurde, wird wieder normal dargestellt, wenn ein anderer Knopf (Button) angeklickt wird. Der angeklickte Knopf erscheint nun revers.
5	32	LASTOB: Mit diesem Flag muß das letzte Objekt einer Objektstruktur gekennzeichnet werden. Bei Nichtbeachtung stürzt der Rechner evtl. ab. Das Resource-Construction-Set nimmt die Kennzeichnung selbst vor. Sie brauchen daher in der Regel nicht darauf zu achten.
6	64	TOUCHEXIT: Die Kontrolle wird vom AES an das Programm zurückgegeben, wenn die Maustaste gedrückt wird. AES wartet nicht auf das Loslassen der Taste. Für eine einwandfreie Funktion darf Bit 0 nicht gesetzt sein.
7	128	HIDETREE
8	256	INDIREKT

Durch Addition der Werte lassen sich auch mehrere Flags gleichzeitig setzen. Beachten Sie dabei bitte das oben Gesagte.

Es geht weiter, mit Wort 5 der Objektstruktur.

Wort 5 In Drop-down-Menüs werden manchmal Einträge, die aktiv sind, mit einem Häkchen versehen. Sie haben sie schon im ersten Teil des Kapitels kennengelernt. Ob zum Beispiel ein Eintrag so gekennzeichnet wird, bestimmt ein Bit in Wort 5 der Parameterliste. Es handelt sich um das Statusregister. Auch in diesem Register dürfen für ein Objekt mehrere Bits gesetzt sein.

Zur Bedeutung der einzelnen Bits:

Bit	Dez. Wert	Bedeutung
0	1	SELECTED: Das Objekt wird revers dargestellt, also selektiert.
1	2	CROSSED: In das Objekt wird ein Kreuz geschrieben.

Bit	Dez. Wert	Bedeutung
2	4	CHECKED: Es wird ein Häkchen gezeichnet, zum Beispiel bei Drop-down-Menüs.
3	8	DISABLED: Das Objekt wird hell gezeichnet, ist also nicht aktivierbar.
4	16	OUTLINED: Um das Objekt wird ein Rand gezeichnet.
5	32	SHADOWED: Das Objekt wirft einen Schatten.

Durch Addition der einzelnen Werte lassen sich auch mehrere Merkmale setzen. Wir kommen zur Beschreibung der letzten Wörter des Feldes.

Wort 6 und **7** Die Bedeutung dieser Wörter der Parameterliste haben wir bereits bei der Beschreibung der Objekttypen erklärt.

Wort 8 und **9** Um ein Objekt zeichnen zu können, muß das AES wissen, wo es auf dem Bildschirm erscheinen soll. In Wort 8 steht die X-Koordinate und in Wort 9 die Y-Koordinate. Die Koordinaten beziehen sich, außer beim Wurzelobjekt, nicht auf den Bildschirmursprung, sondern stellen die Position des Objekts relativ zum Objekt der höheren Hierarchie dar. Ein untergeordnetes Objekt darf nicht größer sein als das ihm übergeordnete. Alles, was größer ist, wird vom AES nicht gezeichnet.

Wort 10 und **11** In Wort 10 steht die Breite und in Wort 11 die Höhe des Objekts. Alle Angaben werden in Pixeln angegeben.

Sie haben es fast geschafft. Wir sind Ihnen aber noch die Erklärung für weitere wichtige Strukturelemente schuldig. Eines ist die BITBLK-Struktur, das andere die TEDINFO-Struktur. Bei der Verwendung entsprechender Objekttypen besteht ein Zeiger, gebildet von Wort 6 und 7, auf eben diese Strukturen. Sie beschreiben dann das Objekt noch näher. Wir beginnen mit der BITBLK-Struktur. Sie beschreibt ein Grafikobjekt und besteht, ohne Grafikdaten, aus weiteren sieben Wörtern. Hier die Belegung:

Wort	Bedeutung
0–1	Zeiger auf eine Adresse, ab der die Grafik bitweise abgelegt ist. Jeder Eintrag hat Wortlänge. Jedes gesetzte Bit entspricht einem gesetzten Punkt.
2	Gibt die Breite der Grafik in Bytes an.
3	Gibt die Höhe der Grafik in Linien an.

Wort	Bedeutung
4	Erste X-Position in Bits, d.h., ist der erste Bit-Eintrag auch gleich der erste X-Bit-Eintrag, so wird hier Null übergeben (dies ist der Regelfall). Beginnt der erste X-Eintrag erst beim zweiten Bit, so muß Bit 14 gesetzt werden.
5	Das gleiche wie Wort 4 für Y.
6	Farbe der Grafik

Kommen wir zur TEDINFO-Struktur. In vielen Dialogboxen kann man Texte eingeben, daher nennt man sie auch Formulare. AES gibt dem Programmierer nun über die TEDINFO-Struktur ein mächtiges Werkzeug in die Hand. Sie beschreibt nämlich, welche Zeichen an welcher Stelle des Textfeldes eingegeben werden dürfen, die Farbe und Länge des Textes usw. Jede Struktur dieser Art hat daher entsprechend viele Einträge, sie umfaßt 14 Wörter. Die Belegung:

Wort	Bedeutung
0–1	Es handelt sich um einen Zeiger, der auf den auszugebenden Text gerichtet ist. Der String muß unbedingt mit einem Nullbyte abgeschlossen sein. Der GFA-Befehl »CHAR« nimmt Ihnen diese Arbeit ab. Ist der Text veränderbar, ist der geänderte Text später ebenfalls ab dieser Adresse (abgeschlossen durch ein Nullbyte) zu finden.
2–3	Zeiger auf einen Text, mit dem der auszugebende oder veränderbare Text gemischt wird. Jedes Zeichen des Haupttextes, das veränderbar sein soll, muß in diesem String das Zeichen »_« erhalten. Auch dieser String muß mit einem Nullbyte abgeschlossen werden.
4–5	Zeiger auf einen String, der angibt, welche Zeichen an welcher Stelle zugelassen sind. Der Programmierer kann zwischen verschiedenen Möglichkeiten wählen. Es bedeutet: 9 Ziffern 0–9 zulassen A Großbuchstaben A–Z und Leerzeichen N Zeichen 0–9 A–Z und Leerzeichen F TOS-Filenamenzeichen und ?, *, : P TOS-Filenamenzeichen und *, ?, \, : X alle Zeichen sind zugelassen a Zeichen a–z, A–Z und Leerzeichen

Wort	Bedeutung
	n Zeichen a–z, A–Z, 0–9 und Leerzeichen p TOS-Filenamenzeichen und \, : Der String muß mit einem Nullbyte abgeschlossen werden.
6	Dieses Wort bestimmt den Zeichensatz, in dem der Text dargestellt werden soll. 3 heißt normal hoher Zeichensatz, 5 schaltet auf den kleineren Satz um.
7	Dieses Wort ist ohne Bedeutung
8	Dient zum Formatieren des Textes 0 = linksbündig, 1 = rechtsbündig, 2 = zentriert
9	Gibt die Farbe des Textes an
10	Dieses Wort ist ohne Bedeutung
11	Bei BOXTEXT und FBOXTEXT gibt dieses Byte die Dicke des Randes des Rechtecks an. 1 bis 128 Randdicke nach innen, –1 bis –127 Randdicke nach außen.
12	Gibt die Länge des Textes einschließlich des Nullbytes an
13	Gibt die Länge des Mischtextes an, siehe Wort 2 und Wort 3. Die Länge ergibt sich aus Textlänge plus Nullbyte.

Bevor Sie das Buch nun zuklappen, weil es Ihnen zu langweilig wird, beginnen wir mit dem Erstellen eines Programms, das eine recht einfache Dialogbox abfragt. Viele der obenstehenden Informationen benötigen Sie bei einfacheren Anwendungen nicht. Unser Buch versteht sich aber auch als Nachschlagewerk. Es soll Sie, in einem gewissen Rahmen, deshalb auch nicht bei der Realisierung komplexerer Anwendungen im Stich lassen. Starten Sie nun bitte unser Demoprogramm. Es trägt den Namen 3_2.GFA. Nach dem Start haben Sie folgenden Bildschirmaufbau:

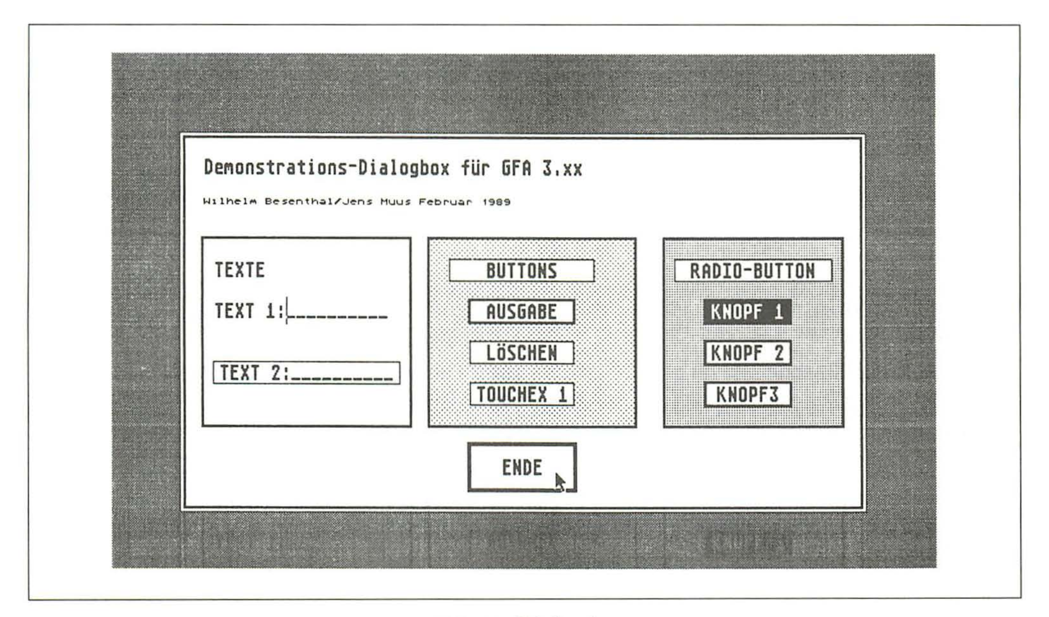


Bild 3.2: Dialogbox

Die Programmerklärung

In den Feldern »Text 1« und »Text 2« können Sie beliebige Texte eingeben. Löschen geht entweder über die Tastatur oder über den Button LÖSCHEN. Der Button ANZEIGEN zeigt Ihnen die Texte und den gewählten Radio-Button des rechten Teils der Box an. Der Button TOUCHEXIT dient lediglich der Demonstration dieses Merkmals.

Beachten Sie bitte, daß das Tilde-Zeichen eine Rückgabevariable ersetzt. Beim Einsatz einer Variablen sollte diese auf jeden Fall vom Typ *Long-Integer* sein. Ist ihr Inhalt nach der Rückkehr größer als 0, verlief der AES-Aufruf ohne Fehler, ist er dagegen gleich 0, trat ein Fehler auf.

Das Programmlisting:

```
'
' ************************** BEMU 89 ***************************
' ********************** Dialogbox-Demo ************************
' *************** Programm aus dem GfA 3.0 Buch ****************
' ************ Von Wilhelm Besenthal & Jens Muus ***************
' ********* erschienen im Markt & Technik Verlag 1989 *********
' **************************************************************
'
```

```
GOSUB definitionen
GOSUB rsrc_load
GOSUB objekt_nr
GOSUB dialogbox_zeichnen
GOSUB box_abfrage
```

Auch dieses Programm besteht nur aus Unterprogrammen, die nacheinander aufgerufen werden. Im ersten legt das Programm einige wichtige Dinge für den Programmablauf fest.

```
'
PROCEDURE definitionen
  RESERVE -1024                      !1024 Byte für RSC-Datei
  ON ERROR GOSUB ende                !Bei Fehler Programm beenden
  DEFFILL 1,2,4                      !Füllmuster für Desktop
  BOUNDARY 0                         !Kein Rand bei PBOX usw.
  PBOX 0,0,639,399                   !Desktop füllen
  pfad$=DIR$(0)                      !Aktuellen Pfad ermitteln
  rsrc_name$=pfad$+"\3_2.RSC"        !Name der RSC-Datei
RETURN
```

Die Dialogbox liegt als Resource-Datei auf der Diskette vor. Die Datei heißt: 3_2.RSC. Da sie später in den Rechner geladen wird, müssen wir für sie Speicherplatz reservieren. Die nächsten Programmzeilen erzeugen ein Desktop. Um das Programm möglichst flexibel zu halten, wird der aktuelle Suchpfad des aktuellen Laufwerks erfragt. Suchpfad und Name der Resource-Datei übergibt das Programm an die Variable »rsrc_name$«.

Die nächste Prozedur lädt die Datei in den Rechnerspeicher.

```
PROCEDURE rsrc_load
  IF RSRC_LOAD(rsrc_name$)=0         !Wenn Datei nicht vorhanden
    ALERT 1," RSC-DATEI | Nicht gefunden!",1," OK ",bu&
    GOSUB ende
  ENDIF
RETURN
```

Zum Einladen der RSC-Datei bedient man sich des folgenden AES-Aufrufs:

R&=RSRC_LOAD(Name$)

Enthält die Variable »R&« nach der Rückkehr der Funktion einen Wert verschieden 0, war der Aufruf erfolgreich, die Strukturen stehen jetzt im Speicher. Ist der Rückgabewert dagegen 0, dann ging etwas schief. In den meisten Fällen stimmt der Suchpfad nicht oder die Datei ist gar nicht vorhanden. Unser Programm meldet sich dann mit einer entsprechenden Meldung bei Ihnen.

```
PROCEDURE objekt_nr
  '
  ' Hier werden den Variablen die Objektnummern zugewiesen
  '
  LET tree1&=0                              !RSC_TREE
  LET knopf1&=2                             !Obj in #0
  LET knopf2&=3                             !Obj in #0
  LET knopf3&=4                             !Obj in #0
  LET text1&=8                              !Obj in #0
  LET text2&=9                              !Obj in #0
  LET ausgabe&=13                           !Obj in #0
  LET loeschen&=14                          !Obj in #0
  LET touchex1&=15                          !Obj in #0
  LET ende&=6                               !Obj in #0
RETURN
```

Mit dem von GFA gelieferten *Resource-Construction-Set* lassen sich Dateien mit der Extension ».LST« erzeugen. Die Objektnamen, die Sie den Objekten bei der Erstellung des Resource gegeben haben, werden beim Speichern der RSC-Datei als Integer-Variablen deklariert. Ihnen wird dabei gleichzeitig die Objektnummer, die das Construction-Set nach eigenem Gutdünken den Objekten zuteilt, zugewiesen. Das erzeugte ».LST«-File wird später zum Programm nur dazu»gemergt«. Nach Aufruf der obenstehenden Prozedur kann man mit den wesentlich übersichtlicheren Variablennamen weiterarbeiten.

```
PROCEDURE dialogbox_zeichnen
  ' Die Procedure stellt zuerst die Adresse der Baumstruktur fest
  ' und läßt das AES dann die jeweiligen Koordinaten für die
  ' zentrierte Ausgabe berechnen.
  '
  ~RSRC_GADDR(0,tree1&,adr%)                !Adresse der OBJEKT-Struktur
```

Jetzt wird es ernst. Das RSC-File steht im Speicher, uns fehlt aber noch seine Anfangsadresse, da die meisten AES-Aufrufe sie benötigen. Das Programm erfährt sie durch:

~**RSRC_GADDR**(Type,Objnr,Adr%)

Mit Hilfe dieser Funktion läßt sich jede Speicheradresse einer Struktur, also auch Tedinfo usw., feststellen. Von welcher Art die Struktur sein soll, deren Adresse benötigt wird, bestimmt die Variable »Type«. Der Objektbaum, nach seiner Adresse wird wohl in den meisten Fällen gesucht, hat den Typwert 0. Die Variable »Objnr« übergibt der Funktion die Objektnummer. Ist alles glattgegangen (wir gehen davon aus, Sie erkennen das am Tilde-Zeichen vor der Funktion), steht danach in »Adr%« die gesuchte Adresse.

```
  ~FORM_CENTER(adr%,x&,y&,b&,h&)     !Zentrierte Koordinaten
                                     !erfragen
```

Die Dialogbox soll in der Mitte des Bildschirms erscheinen. Wir brauchen uns um die Berechnung der dazu notwendigen Koordinaten nicht zu kümmern, diese Arbeit nimmt uns folgende Funktion ab:

FORM_CENTER(Adr%,X&,Y&,B&,H&)

Der Funktion wird lediglich die Adresse der Baumstruktur übergeben. Sie liefert in den Variablen »X&« bis »H&« die Koordinaten für die zentrierte Ausgabe zurück.

```
GOSUB texte_loeschen             !Editierstrings löschen
```

Die Dialogbox enthält die Möglichkeit, zwei Textzeilen einzugeben. Beim Erstellen des Resource gibt man für jedes erlaubte Zeichen einen Unterstrich »_« an. Damit der Textcursor, nach dem Erscheinen der Box, auch wirklich in der ersten Spalte einer Zeile steht, erfolgt obiger Unterprogrammaufruf. Das Unterprogramm wird später noch beschrieben.

```
~FORM_DIAL(0,0,0,0,0,x&,y&,b&,h&) !Bildschirmbereich reservieren
```

Die Funktion beinhaltet eigentlich vier AES-Aufrufe. Welche Funktion ausgeführt werden soll, entscheidet der erste Parameter. Die Aufrufe in der Übersicht:

FORM_DIAL(P,X1,Y1,B1,H1,X2,Y2,B2,H2)

P — Der Parameter gibt vor, welche Funktion auszuführen ist. Es bedeutet:
0: Reserviert den Bildschirmbereich für die Dialogbox
1: Zeichnet ein sich öffnendes Rechteck
2: Zeichnet ein sich schließendes Rechteck
3: Gibt den reservierten Bildschirmbereich wieder frei und eine Nachricht (über den Message-Buffer) an das Programm.

X1,Y1,B1,H1 — Diese Variablen beschreiben das Rechteck in seiner kleinsten Größe.

X2,Y2,B2,H2 — Diese Variablen beschreiben das Rechteck in seiner größten Größe.

Mit der Programmzeile oben teilen wir dem AES also mit, welchen Bildschirmbereich unsere Dialogbox nach dem Öffnen belegen wird. Es werden nur die Koordinaten der größtmöglichen Ausdehnung der Box übergeben, alle anderen Werte sind 0.

Der nächste Aufruf der Funktion dient lediglich der Optik. Er zeichnet ein größer werdendes Rechteck. Es erreicht zum Schluß die Ausmaße der Box.

```
~FORM_DIAL(1,0,0,0,0,x&,y&,b&,h&) !Größer werdendes Rechteck zeichnen
```

Das Rechteck verschwindet selbstverständlich wieder vom Bildschirm. Es folgt ein erneuter Aufruf des AES. Dieser zeichnet die nachfolgende Box auf den Bildschirm.

~OBJC_DRAW(Adr%,P1,P2,X,Y,B,H)

Es bedeutet:

Adr%	Adresse des Objektbaums.
P1	Objektnummer des ersten zu zeichnenden Objekts.
P2	Anzahl der Ebenen, die gezeichnet werden sollen. Beispiel: Bei P2 = 0 wird nur das Objekt, bei P2 = 1 das Objekt und seine Unterobjekte, bei P2 = 2 das Objekt, seine Unterobjekte und deren Unterobjekte usw. gezeichnet.
X,Y,B,H	Die Variablen beschreiben den Bildschirmteil, in dem das Objekt gezeichnet werden soll.

Die Zeile des Programms sieht so aus, das Unterprogramm hat mit ihr denn auch seine Pflicht und Schuldigkeit getan:

```
  ~OBJC_DRAW(adr%,tree1&,2,x&,y&,b&,h&)  !Dialogbox zeichnen
RETURN
```

Der Aufruf teilt dem AES mit, daß es das Objekt, dessen Objektnummer in der Variablen »tree1&« steht (das ist das Wurzelobjekt, die große Box), mit seinen Unterobjekten (das sind die Boxen und der ENDE-Button) und deren Unterobjekten (das sind die Buttons in den Boxen) zeichnen soll. Die Koordinaten für die Zeichnung stehen in: x&, y&, b& und h&.

Die nächste Prozedur übernimmt die Abfrage der Box und leitet entsprechende Reaktionen, auf Anwenderanforderungen, ein.

```
PROCEDURE box_abfrage
  '
  ' Diese Routine fragt die Aktivitäten des Benutzers innerhalb
  ' der Box ab und leitet dann die entsprechende Reaktion des
  ' Programms ein.
  ' exbu&=> enthält die Nummer des angeklickten Objekts
  DO
    exbu&=FORM_DO(adr%,text1&)
```

Die Überwachung der Box erfolgt vollständig über das AES. Beachten Sie bitte, daß die Abfrage nur über ein EXIT- oder TOUCHEXIT-Objekt beendet werden kann. Wurde vergessen, mindestens ein Objekt durch Setzen des entsprechenden Flags als solches zu kennzeichnen, hilft nur noch die Reset-Taste. Die Beschreibung der Funktion:

Exbu&=FORM_DO(Adresse,Text_objekt)

Exbu& — Das Verlassen der Funktion kann ausschließlich über ein als EXIT-Objekt definiertes Objekt erfolgen. Nach der Rückkehr der Funktion zum Programm befindet sich in der Variablen die Nummer des Objekts, über das die Abfrage beendet wurde.

Adresse — Adresse des Objektbaums.

Text_objekt — Enthält das Formular vom Benutzer editierbare Textelemente, dann muß diese Variable die Objektnummer des Elements enthalten, in der der Cursor nach dem Funktionsaufruf stehen soll. Übergabe von 0 setzt den Cursor auf das erste editierbare Objekt (es zählt die Objektnummer, nicht die Position). Ist kein editierbares Objekt vorhanden, oder soll der Cursor zu Beginn unsichtbar sein, übergibt man –1.

Nachdem das AES die Kontrolle an das Programm zurückgegeben hat, wertet dieses die Eingaben aus. Die Auswertung wird zweckmäßigerweise durch »SELECT ... CASE ... ENDSELECT« vorgenommen.

```
SELECT exbu&
CASE ausgabe&
  '
  ' TEXTE IN STRINGVARIABLEN ÜBERNEHMEN
  '
  t1$="TEXT 1= "+CHAR{{OB_SPEC(adr%,text1&)}}
  t2$="TEXT 2= "+CHAR{{OB_SPEC(adr%,text2&)}}
```

Haben Sie den Button AUSGABE angeklickt, kopiert das Programm mittels der CHAR-Funktion die Eingabestrings in »t1$« und »t2$«. Zum besseren Verständnis betrachten wir den Kopiervorgang in »t1$« näher.

Wie schon bei der Beschreibung der Objektstrukturen erwähnt, verfügt GFA über entsprechende Funktionen, die Zugriff auf eine Objektstruktur haben. In unserem Fall handelt es sich um »OB_SPEC.« Der Funktion übergibt man die Adresse des Baums und die Objektnummer. Sie liefert, als Langwort, den Inhalt der Wörter 6 und 7 der Struktur zurück. Hier handelt es sich um eine Adresse, ab der die TEDINFO-Struktur im Speicher steht. Dem Befehl »CHAR« folgen zwei geschweifte Klammern ({{). Die erste dient der Funktion, die zweite sagt GFA, daß ein indirekter Zugriff erfolgen soll. GFA holt sich deshalb den Inhalt der ersten vier Byte, auf die »OB_SPEC« zeigt. Sie enthalten die Adresse des Textes. Ab dieser Adresse kopiert das Basic alle Zeichen in »t1$«, bis es auf das erste Nullbyte trifft.

```
'
' WELCHER RADIO-BUTTON IST AKTIV?
'
IF OB_STATE(adr%,knopf1&) AND 1
  rb&=1
ELSE IF OB_STATE(adr%,knopf2&) AND 1
  rb&=2
ELSE IF OB_STATE(adr%,knopf3&) AND 1
  rb&=3
ENDIF
rb$="RADIO-BUTTON "+STR$(rb&)+" GEWÄHLT"
```

Als nächsten Schritt überprüft das Programm, welcher Radio-Button aktiv ist. Es kann von den dreien immer nur einer aktiv sein, das Merkmal dieser Button-Art. Ist der Button gefunden, wird der entsprechende Ausgabestring aufgebaut.

```
ALERT 1,t1$+" |"+t2$+"| |"+rb$,1," OK ",bu&
```

Danach stellt das Programm eine Alert-Box dar, in der die erhaltenen Informationen erscheinen.

```
~OBJC_CHANGE(adr%,exbu&,0,x&,y&,b&,h&,0,1)
```

Wenn Sie einen als EXIT-Button definierten Button anwählen, soll er, in den meisten Fällen jedenfalls, nach der inversen Darstellung und entsprechender Reaktion, wieder ein »normales« Aussehen erhalten. Das erreicht das Programm durch Aufruf der Funktion »OBJC_CHANGE«.

Der Aufruf in der Übersicht:

~OBJC_CHANGE(Adr%,Objektnr,0,X,Y,B,H,Neuer_Status,Flag)

Adr%	Adresse des Objektbaums.
Objektnr	Nummer des Objekts, dessen Status geändert werden soll.
0	Reserviert, immer 0.
X,Y,B,H	Innerhalb dieser Koordinaten soll der Status geändert werden.
Neuer_Status	Neuer Status, den das Objekt erhalten soll.
Flag	0 = Objekt nicht neu zeichnen, 1 = Objekt neu zeichnen.

Sind mehrere Bits des Objekt-Statusregisters gesetzt, ändert man zuerst das oder die entsprechenden Bits durch Verknüpfung mit OB_STATE(tree,objekt_nr). Danach folgt ein Aufruf von OBJC_DRAW. Die Funktion beschreiben wir als nächstes.

```
    CASE loeschen&
      GOSUB texte_loeschen
      ~OBJC_DRAW(adr%,text1&,0,x&,y&,b&,h&)  !Textfeld 1 Neu
                                             ! zeichnen
      ~OBJC_DRAW(adr%,text2&,0,x&,y&,b&,h&)  !  "     2  "    "
```

Sollen, durch Anklicken des LÖSCHEN-Buttons, die Texte aus den Textfeldern entfernt werden, ruft das Unterprogramm zunächst ein weiteres Unterprogramm, »texte_loeschen«, auf. Es entfernt zwar die Texte aus dem Speicher, sie werden jedoch nach wie vor angezeigt.

Die AES-Funktion »OBJC_DRAW« sorgt für die Aktualisierung der Dialogbox. Ihre Funktion wird nochmals, allerdings kürzer, beschrieben. Man übergibt der Funktion die Adresse des Objektbaums, die Objektnummer des neu zu zeichnenden Objekts, die Tiefe (hier 0, da nur das Objekt selbst neu gezeichnet werden soll) und die Koordinaten, in denen der neu zu zeichnende Bereich liegt. Als Koordinaten nimmt man einfach die der Dialogbox, das AES richtet's dann schon.

```
    CASE touchex1&
      ALERT 1," Das war | | TOUCHEXIT 1 ",1," OK ",bu&
```

Der Button TOUCHEXIT 1 dient lediglich der Demonstration. Bei der Erstellung des RSC-Files wurde er, wie schon der vorherige, als TOUCHEXIT definiert. Erinnern Sie sich bitte, es handelt sich dabei um ein Flag des Objekts. Ist das Flag gesetzt und das EXIT-Flag nicht, gibt AES die Kontrolle an das Programm beim Drücken der linken Maustaste innerhalb des Objekts sofort zurück. Es wartet nicht auf das Loslassen der Taste wie beim EXIT-Objekt. Das »Betreten« des Objekts mit gedrückter Maustaste hat das gleiche Ergebnis zur Folge.

```
    CASE ende&
      GOSUB ende
    ENDSELECT
  LOOP
RETURN
```

Diese Zeilen erklären sich wohl von selbst.

```
PROCEDURE texte_loeschen
  '
  ' DIESE PROCEDURE LÖSCHT DIE STRINGS IN DER BOX
  '
  CHAR{{OB_SPEC(adr%,text1&)}}=""
  CHAR{{OB_SPEC(adr%,text2&)}}=""
RETURN
```

Ein Text wird, in einer Dialogbox, durch Ersetzen seines ersten Zeichens durch ein Nullbyte gelöscht. AES gibt nämlich immer nur die Zeichen aus, die vor einem Nullbyte stehen.

Der Befehl CHAR greift wieder indirekt auf den Speicher zu. Wie das funktioniert, haben wir schon beschrieben.

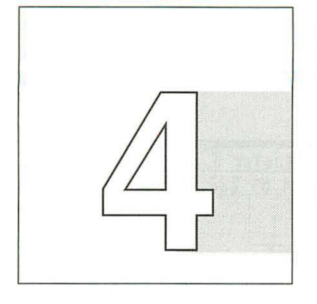
Es folgt die letzte Routine des Programms:

```
PROCEDURE ende
  ~FORM_DIAL(2,0,0,0,0,x&,y&,b&,h&) !Kleiner werdendes Rechteck
                                    !zeichnen
  ~FORM_DIAL(3,0,0,0,0,x&,y&,b&,h&) !Bildschirm restaurieren
  ~RSRC_FREE()
  RESERVE
  END
RETURN
```

Der erste FORM_DIAL-Aufruf dient nur der Optik. Er zeichnet ein schrumpfendes Rechteck, dessen Anfangsgröße mit den Variablen »x&« bis »h&« und dessen Endgröße durch 0,0,0,0 definiert ist. Sehr wichtig ist der zweite FORM_DIAL-Aufruf. Er baut den Hintergrund der Box wieder auf und sendet danach eine Redraw-Message über den Message-Buffer an das Programm. Sie erfahren dazu noch mehr im Kapitel über die Fensterverwaltung.

Hinweis: Der Hintergrund wird grundsätzlich mit dem grauen Muster des Desktops aufgebaut. Bei Verwendung einer anderen Hintergrundfarbe muß das Programm ihn selber wieder restaurieren. Überdeckte die Box Randbereiche von Fenstern, werden diese vom AES neu gezeichnet. Für die Wiederherstellung der Fensterinhalte ist ebenfalls das Programm zuständig.

Das nächste Kapitel behandelt die Fensterprogrammierung unter GEM. Wir hoffen, daß Sie dabei sind.



4 Die Fensterverwaltung

Die Ausgaben eines Programms unter GEM erfolgen in den meisten Fällen in einem oder mehreren Fenstern. Wie ein Fenster aussieht, brauchen wir Ihnen sicher nicht mehr zu erklären, Sie reizt bestimmt viel mehr die Programmierung dieser wichtigen Hilfsmittel.

Schwierige Themen lassen sich meistens anhand eines praxisnahen Beispiels besser erklären. Das sagten wir uns jedenfalls, und so entstand das Programm 4.GFA. Nach den auch bei diesem Thema unumgänglichen Erklärungen der Grundfunktionen des AES erläutern wir diese dann nochmals im Zusammenhang mit dem Listing. Sie finden das Programm unter dem Namen 4.GFA auf der dem Buch beiliegenden Diskette. Dem Programm lagen folgende Ideen zugrunde:

Das Programm sollte:

- einen beliebig großen Bildschirm im Hintergrund verwalten können,
- im aktuellen Fenster das Zeichnen von Rechtecken ermöglichen,
- die Bedienung der Fenster über alle Fensterattribute ermöglichen, die das GEM bereithält,
- alle unter GEM verfügbaren Fenster öffnen können,
- und, der wohl wichtigste Punkt, so programmiert werden, daß die Routinen schnell den eigenen Bedürfnissen angepaßt werden können und dabei trotzdem durchschaubar bleiben.

Ob gerade der letzte Punkt erfüllt wird, entscheiden Sie, lieber Leser. Sie erhalten mit diesem Programm kein Zeichenprogramm, seine Programmierung überlassen wir Ihnen, sondern eine Unterprogrammsammlung, die die Fensterverwaltung sehr vereinfacht. Starten Sie jetzt einmal das Programm. Es bietet sich Ihnen danach das folgende Bild:

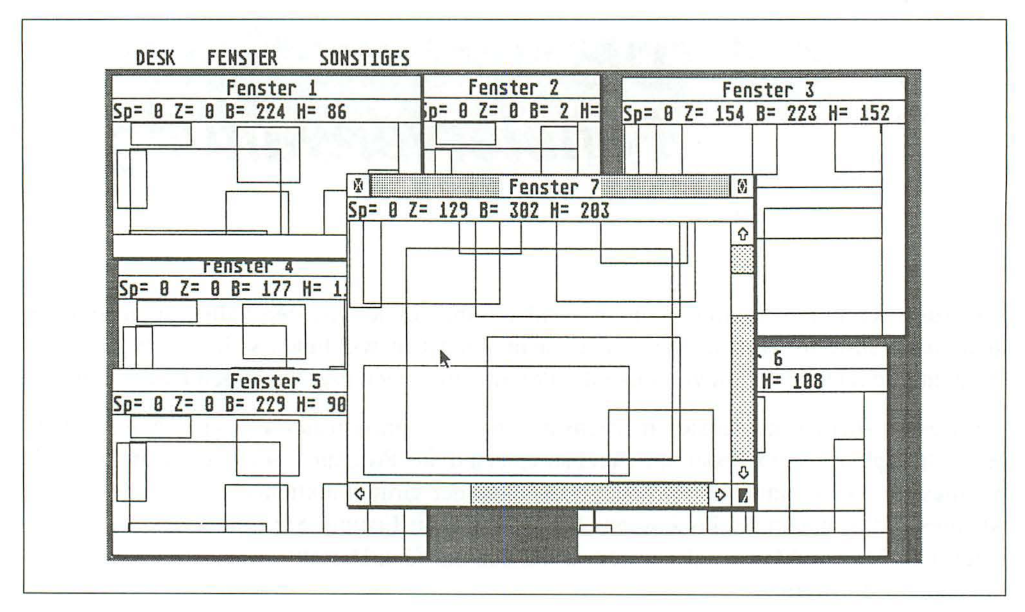


Bild 4.1: Fensterdemo

War der Start nicht erfolgreich, verfügt Ihr Rechner über zu wenig freien Speicherplatz. Sie müssen dann die Größe des Zeichenbildschirms verkleinern. Die Einstellungen werden in der ersten Prozedur des Programms vorgenommen und später noch genau beschrieben. Es folgt eine kurze Bedienungsanleitung.

Sie können bis zu sieben Fenster über den entsprechenden Menüpunkt öffnen. Alle Fenster greifen auf denselben, im Hintergrund angelegten Grafikbildschirm zu. Der Speicher wurde von uns in mehr Zeilen und Spalten aufgeteilt, als in einem Fenster in Maximalgröße darstellbar sind. Horizontales und vertikales Scrollen geschieht mit Hilfe der entsprechenden Slider und Pfeilsymbole am Fensterrand. In der Infozeile befindet sich immer die aktuelle Spalte und Zeile der linken oberen Ecke des Fensters, bezogen auf den Grafikbildschirm. In ihr sind außerdem noch die Breite und Höhe des Fensters zu sehen. Da alle Fenster den gleichen Speicherbereich anzeigen, kann man sich so sehr einfach verschiedene Ausschnitte anzeigen lassen. Alle übrigen Optionen eines Fensters wie Schließen, Verändern der Größe, Verschieben und so weiter beherrscht das Programm ebenfalls.

Im aktuellen Fenster ist es wie gesagt möglich, Rechtecke zu zeichnen.

- Zuerst legt man die linke obere Ecke durch entsprechende Positionierung des Mauscursors und Klicken mit der linken Maustaste fest.
- Bewegen der Maus erzeugt ein entsprechendes Rechteck. Klicken mit der rechten Maustaste zeichnet das Rechteck endgültig.

Obwohl alle Fenster auf denselben Grafikbildschirm zugreifen, werden neue »Zeichnungen« in die evtl. vorhandenen anderen Fenster nicht sofort übernommen. Das Neuzeichnen aller geöffneten Fenster erreicht man durch Wahl des entsprechenden Menüpunktes. Ein anderer Menüpunkt ermöglicht das Löschen des gesamten Grafikbildschirms. Soviel zur Bedienung, wir beschreiben jetzt die Grundzüge der Fensterprogrammierung.

4.1 Grundlagen

Wie bei vielen Dingen des täglichen Lebens kommt man auch bei der Fensterverwaltung ohne Grundlagen nicht aus. Sie sollten diesen Teil besonders gut durchlesen, das Verstehen des Listings fällt Ihnen dann bestimmt wesentlich leichter.

Wir werden im folgenden sehr oft den Begriff Handle verwenden, deshalb erklären wir ihn sofort. Jedes Fenster, das vom AES angelegt wird, erhält von ihm eine Zuordnungsziffer. Diese Ziffer wird dem Programm nach dem Anlegen (wie man das veranlaßt, kommt noch) zurückgegeben. Eben diese Kennziffer ist nichts anderes als das schon angesprochene *Handle*. Das Programm sollte es sich immer merken, da alle Fensterfunktionen des AES bei einem Aufruf nach ihm verlangen. Wie man das Handle eines unbekannten Fensters erfährt, zeigen wir Ihnen bei der Programmerklärung.

Bevor mit Fenstern gearbeitet werden kann, muß sich der Programmierer über das Aussehen im klaren sein. Wie Sie wissen, kann ein Fenster eine ganze Reihe Bedienelemente besitzen. Jedes Element kann vorhanden oder nicht vorhanden sein. Bestimmt wird das mit einer sogenannten *Komponentenmaske*, die beim Anlegen eines Fensters der entsprechenden AES-Funktion übergeben wird.

Die Maske besteht aus 12 Bit. Jedes gesetzte Bit repräsentiert ein vorhandenes Bedienelement. Die Bedeutung der Bits im einzelnen:

Bit	Wert	Bedeutung
0	1	Das Fenster hat eine Titelzeile mit Namen.
1	2	Das Fenster besitzt ein Löschfeld.
2	4	Das Fenster besitzt ein Full-Feld.
3	8	Das Fenster hat ein Bewegungsfeld.
4	16	Das Fenster erhält eine Infozeile.
5	32	Es kann in der Größe verändert werden.
6	64	Am rechten Fensterrand befindet sich ein Pfeil nach oben.
7	128	Am rechten Fensterrand befindet sich ein Pfeil nach unten.

Bit	Wert	Bedeutung
8	256	Am rechten Fensterrand befindet sich ein vertikaler Schieber.
9	512	Am unteren Fensterrand gibt es einen Pfeil nach links.
10	1024	Am unteren Fensterrand gibt es einen Pfeil nach rechts.
11	2048	Das Fenster besitzt am unteren Rand einen horizontalen Schieber.

Addition entsprechender Werte setzt mehrere Bedienelemente. Soll ein Fenster über alle Attribute verfügen, übergibt man die Zahl 4095; rechnen Sie ruhig nach.

Nachdem die Komponenten festgelegt sind, ist als nächstes die Überlegung angebracht, wie groß das später erscheinende Fenster sein darf. Es ist nämlich nicht immer die maximal mögliche Größe gewünscht, man denke nur zum Beispiel an Zeichenprogramme, die über eine Randleiste verfügen. Es gibt zwei Größen, zwischen denen es zu unterscheiden gilt: Die erste ist die Gesamtgröße des Fensters, d.h. einschließlich der Randbereiche (Titelzeile, Slider usw.), die zweite der eigentliche Arbeitsbereich (nur in ihm sollten Ausgaben erfolgen). Wir gehen bei allen folgenden Erklärungen davon aus, daß die Gesamtausmaße eines Fensters die volle Größe des Bildschirms (abzüglich der Menüleiste) annehmen können.

Viele Programme sollen in allen Bildschirmauflösungen problemlos laufen. Damit nun ein Fenster nicht größere Ausmaße annehmen kann als der eigentliche Bildschirm, ist zunächst das Feststellen der Bildschirmauflösung wichtig. Bedienen Sie sich dabei bitte nicht der entsprechenden TOS-Funktion, Besitzer eines Monitors höherer Auflösung werden es Ihnen danken. Wählen Sie einfach folgende AES-Funktion:

Erg=WIND_GET(Handle,Fnr,X,Y,B,H)

Mit Hilfe der Funktion läßt sich, in Abhängigkeit von der Funktionsnummer »Fnr«, eine ganze Menge über das Fenster mit der in Handle angegebenen Nummer erfahren. Die Rückgabewerte stehen immer in den Variablen »X, Y, B« und »H«. X und Y enthalten die jeweiligen Koordinaten der linken oberen Ecke, B und H geben die Breite und Höhe eines Bereiches an. In der Variablen »Erg« findet man im allgemeinen nach dem AES-Aufruf eine Fehlernummer. Ist sie null, ging etwas schief. Die Variable läßt sich auch durch das Tilde-Zeichen (~) ersetzen.

Folgende Funktionsnummern sind möglich:

Fnr	Rückgabe von...
4	X-, Y-Koordinate sowie Breite und Höhe des Arbeitsbereiches.
5	X-, Y-Koordinate sowie Breite und Höhe der Gesamtgröße.
6	X-, Y-Koordinate sowie Breite und Höhe der Gesamtgröße des vorherigen Fensters.
7	X-, Y-Koordinate sowie Breite und Höhe der möglichen, vorher festgelegten, Gesamtgröße.
8	Position des horizontalen Sliders (Schiebers). Es sind Werte von 0 bis 1000 möglich.
9	Position des vertikalen Sliders. Es sind Werte von 0 bis 1000 möglich.
10	Handle des aktiven Fensters.
11	X-, Y-Koordinate sowie Breite und Höhe des ersten Rechtecks der Rechteckliste (Erklärung siehe unten).
12	X-, Y-Koordinate sowie Breite und Höhe des nächsten Rechtecks der Rechteckliste (Erklärung siehe unten).
15	Größe des horizontalen Schiebers relativ zur Größe des Kastens.
16	Größe des vertikalen Schiebers relativ zur Größe des Kastens.

Durch Erg=WIND_GET(0,4,X,Y,B,H) erhalten wir die Gesamtgröße des Desktops (Handle=0) zurück.

Die mögliche Gesamtgröße haben wir vom AES erhalten. Wie erfährt man nun die Größe des Arbeitsbereichs? Die eben beschriebene Funktion scheidet aus, da noch kein Fenster angelegt wurde und somit noch kein Handle zur Verfügung steht. Die Lösung bietet ein anderer Aufruf des GEM:

Erg=WIND_CALC(Fnr,Komp,X,Y,B,H,R_x,R_y,R_b,R_h)

Die Variablen im einzelnen:

Erg	Ergebnis des Aufrufs; wenn 0, trat ein Fehler auf.
Fnr	Funktionsnummer; wird 0 übergeben, berechnet die Funktion die Gesamtgröße des Fensters, bei 1 berechnet sie den Arbeitsbereich.
Komp	Komponentenmaske

X – H Diese Koordinaten werden übergeben.

R_x – R_h Diese Koordinaten erhält man zurück.

Da in unserem Beispiel die Gesamtgröße feststeht, wählen wir die Funktionsnummer 1 und übergeben ihr die bekannten Koordinaten in den Variablen »X – H«. Wir erhalten in »R_x – R_h« die Größe des Arbeitsbereichs zurück.

Variationsmöglichkeiten:

- Man muß die Größe des Arbeitsbereichs nicht unbedingt vor dem Anlegen wissen. Ist das Fenster erst einmal angelegt (wird gleich beschrieben), liefert uns GEM auf Verlangen diese durch den entsprechenden WIND_GET-Aufruf.
- Soll das Fenster nie in seiner größtmöglichen Größe erscheinen, so ruft man »WIND_CALC« mit den gewünschten Größenkoordinaten auf.
- Wünscht man eine bestimmte Arbeitsbereichsgröße, selbstverständlich im erlaubten Rahmen, ruft man »WIND_CALC« unter Angabe dieser Koordinaten und Verwendung der Funktionsnummer 0 auf.

Die Vorarbeiten sind erledigt, gehen wir daran, das imaginäre Fenster auf den Bildschirm zu bringen. Wir erwähnten schon mehrfach, daß ein Fenster »angelegt« werden muß. Diesen Vorgang nimmt uns wiederum das AES über eine entsprechende Funktion ab.

H%=WIND_CREATE(Komp,Gx,Gy,Gb,Gh)

Die Variablen:

H% Das Handle (die Fensternummer), unter dem von nun an angesprochen werden kann. Ist keins mehr frei, erhält man den Wert 65535 zurück. Aus diesem Grund muß die Variable vom Typ Long-Integer oder Float sein.

Komp Enthält die Komponentenmaske.

Gx–Gh Die maximale Gesamtgröße, die das Fenster später einnehmen kann.

Nach dem Aufruf ist das Fenster noch nicht zu sehen. Obige Funktion teilt dem AES nämlich lediglich mit, daß der Anwender gedenkt, ein Fenster zu öffnen, das die in den Variablen X–H angegebenen maximalen Ausmaße annehmen kann. Das AES »merkt« sich die maximalen Maße und gibt ein noch freies Handle oder die Werte 65535 bzw. –1 zurück.

Achtung: Vergewissern Sie sich durch entsprechende Programmierung, ob allen Aufrufen, die nach einem Handle verlangen, immer nur ein gültiges (kleiner 8) übergeben wird. Der Rechner läßt sich sonst in 99% der Fälle nur noch durch Drücken der »Reset«-Taste wieder zum Leben erwecken.

Die nächste Funktion bringt das Fenster endlich auf den Bildschirm.

Erg=WIND_OPEN(Handle%,X,Y,B,H)

Die Variablen:

Erg	Ist das Ergebnis 0, dann trat ein Fehler auf. Vielleicht wurden in den Variablen »X–H« größere Werte übergeben als vorher bei »WIND_CREATE« festgelegt.
Handle%	Das Handle, das uns »WIND_CREATE« zurücklieferte.
X–H	Die Koordinaten, an denen das Fenster auf dem Bildschirm erscheinen soll. Sie müssen innerhalb des Bereichs der bei »WIND_CREATE« verwendeten liegen.

Nach erfolgreicher Ausführung sieht man jetzt ein Fenster auf dem Bildschirm. Wir wollen mit ihm zunächst keine weiteren Aktionen vornehmen, sondern lassen es gleich wieder verschwinden. Dazu dient der AES-Aufruf:

~WIND_CLOSE(Handle)

Diese Funktion entfernt das Fenster vom Bildschirm, das AES »vergißt« es aber nicht. Das heißt, Sie können es jederzeit erneut mit »~WIND_OPEN(Handle)« auf den Bildschirm bringen. Der folgende Aufruf löscht das Fenster aus dem Gedächtnis des AES:

~WIND_DELETE(Handle)

Diese AES-Funktion bewirkt, daß das AES das Handle zur erneuten Verwendung wieder freigibt. Das vormals mittels »WIND_CREATE« angelegte Fenster existiert nicht mehr. Wird irgendwann einmal ein neues Fenster angelegt, so wird das freigegebene Handle neu vergeben.

Achtung:

- Führen Sie in Ihrem Programm unbedingt eine Liste über die geöffneten bzw. geschlossenen und entfernten Handles. GFA entfernt nämlich nach dem Programmende die Fenster nicht selbständig aus der AES-Liste. Bei erneutem Programmlauf haben Sie dann entweder keine oder nur noch wenige Handles zur Verfügung, sprich, es lassen sich nicht mehr alle Fenster öffnen.
- Halten Sie immer die Reihenfolge »WIND_CLOSE« und dann erst »WIND_DELETE« ein. Mißachtung straft der Rechner oft mit »Tiefschlaf«.
- Versuchen Sie nie, ein Fenster zu schließen oder zu entfernen, dessen Handle nicht vom AES vergeben wurde. Der nächste »Absturz« ist vorprogrammiert.

Soviel zum Öffnen und Schließen der Fenster. Die oben beschriebene Lösung ist eine der leichtesten zur Verwaltung von Fenstern. Sind nämlich alle Fensterattribute gesetzt, läßt einem die Programmierung von Reaktionen auf Anwenderanforderungen schnell graue Haare sprießen. Wir wollen Ihnen an dieser Stelle aber wirklich nicht den Mut nehmen, sondern ganz im Gegenteil Ihr Interesse am Kommenden wecken.

4.2 Der Message-Buffer

Wir haben den Message-Buffer schon im vorherigen Kapitel erwähnt, jetzt nehmen wir seine Funktion und seinen Aufbau genauer unter die Lupe.

Nehmen wir an, daß sich ein Fenster auf dem Bildschirm befindet, das über alle Fensterattribute verfügt. Innerhalb des Arbeitsbereichs soll es dem Anwender möglich sein zu zeichnen. Das Programm hätte nun viel zu tun und wäre daher auch entsprechend langsam, wenn es nicht nur die Aktivitäten des Anwenders beim Zeichnen, sondern ebenfalls noch laufend die Mausposition auf das Betreten der Randelemente überwachen müßte. Das sagten sich wohl auch die Programmierer des GEM, und darum übernimmt die Überwachung das AES, wenn es dazu aufgefordert wird. Den Wunsch auf Überwachung kann das Programm unter GFA-Basic über einen »EVNT_MESSAGE-« bzw. »EVNT_MULTI-« oder einen ON MENU-Aufruf kundtun. Bei ersterem muß eine Nachricht zugelassen, bei der zweiten Lösung durch »ON MENU MESSAGE GOSUB« ein Unterprogrammaufruf definiert sein. Der wesentliche Unterschied zwischen beiden Lösungswegen besteht darin, daß »EVNT_MESSAGE« und »EVNT_MULTI« auf das Eintreffen einer Nachricht warten, »ON MENU« jedoch lediglich überprüft, ob eine Nachricht vorliegt, und wenn nicht, mit der Abarbeitung fortfährt.

Eine Nachricht besteht beim GEM immer aus einer Nachrichtennummer und der Rückgabe eines oder mehrerer Parameter, die die Nachricht näher beschreiben. Sowohl die Nummer als auch die Parameter sind im Message-Buffer zu finden. Dieser Buffer muß nur bei der Verwendung von »EVNT_MESSAGE« bzw. »EVNT_MULTI« eingerichtet werden und besteht aus einem Feld, das 8 Wörter breit ist, das entspricht 16 Byte. Bei Verwendung von »ON MENU« wird der Buffer durch die Variablen »MENU(1)« bis »MENU(8)« dargestellt. Die folgenden Nachrichten können auftreten (in der Aufstellung bedeutet z.B. WORT 4 des Message-Buffers gleich »MENU(4)«). Die Nummer der Nachricht steht immer im Wort 1 des Buffers.

Nachricht Nr.	Weitere Einträge und ihre Bedeutung
10	Es wurde ein Menüpunkt des Pull-down-Menüs ausgewählt. WORT 4: Nummer des Titels WORT 5: Nummer des Menüpunkts
20	Bereich neu zeichnen WORT 4: Fensternummer WORT 5: X-Koordinate des Bereichs WORT 6: Y-Koordinate des Bereichs WORT 7: Breite des Bereichs WORT 8: Höhe des Bereichs
22	Das Löschfeld wurde angeklickt WORT 4: Fensternummer

Nachricht Nr.	Weitere Einträge und ihre Bedeutung
23	Das Vollfeld wurde angeklickt WORT 4: Fensternummer
24	Einer der vier Pfeile wurde angeklickt WORT 4: Fensternummer WORT 5: Pfeil, der angeklickt wurde, es bedeutet: 0: Seite nach oben 1: Seite nach unten 2: Zeile nach oben 3: Zeile nach unten 4: Seite nach links 5: Seite nach rechts 6: Spalte nach links 7: Spalte nach rechts
25	Der horizontale Schieber wurde verändert WORT 4: Fensternummer WORT 5: Position des Schiebers (0–1000)
26	Der vertikale Schieber wurde verändert WORT 4: Fensternummer WORT 5: Position des Schiebers (0–1000)
27	Die Fenstergröße soll geändert werden WORT 4: Fensternummer WORT 5: X-Koordinate WORT 6: Y-Koordinate WORT 7: Neue Breite WORT 8: Neue Höhe
28	Das aktuelle Fenster soll verschoben werden WORT 4: Fensternummer WORT 5: Neue X-Koordinate WORT 6: Neue Y-Koordinate WORT 7: Breite des Fensters WORT 8: Höhe des Fensters
29	Es wurde ein Fenster aktiviert. WORT 4: Fensternummer
40	Es wurde ein Accessory aktiviert WORT 4: ID-Nummer des Accessorys
41	Das Programm wurde abgebrochen WORT 4: ID-Nummer des Accessorys

AES läßt dem Programm also eine ganze Menge Nachrichten zukommen. Wie ein Programm darauf zu reagieren hat, erfahren Sie jetzt, bei der Erklärung des Programms 4.GFA. Ein Studium lohnt sich, bewahrt es Sie doch vor manch mehr oder weniger folgenschwerem Fehler bei der Realisierung eigener Projekte.

4.3 Fensterprogrammierung im Programm

Unser Programm kann einen sehr großen Bildschirm, der im Hintergrund angelegt wird, verwalten. Seine maximale Größe beträgt 32767 mal 32767 Punkte. Innerhalb dieses Bereichs ist die wählbare Größe nur vom Speicherausbau Ihres ST abhängig, das mögliche Maximum ist noch nicht einmal mit einem Mega ST4 erreichbar. Wir erwähnten schon, daß jedes Fenster einen beliebigen Ausschnitt des Grafikbildschirms anzeigen kann. Für den Transport der Grafikdaten aus dem Speicher auf den Bildschirm und umgekehrt zeichnet eine LINE A-Routine verantwortlich. Sie soll im Zusammenhang vorgestellt werden, denn im Programm finden Sie einen Teil von ihr, der in unserem Fall vordefiniert werden konnte, in der Prozedur Definitionen. Die restlichen Teile sind in den Prozeduren die den eigentlichen Transport veranlassen, vorhanden.

Vor dem Aufruf der BITBLT-Routine muß ein *Long-Integer*-Feld mit mindestens 25 Feldelementen dimensioniert werden. Die einzelnen Elemente werden vorbelegt, danach folgt der Befehl:

BITBLT Feldname%()

Sollen Bildschirmausschnitte in den Speicher kopiert werden, so empfiehlt sich das vorherige Abschalten der Maus. Man entgeht so dem Kopieren des Mauscursors.

Die Einträge des Feldes:

`bb%(0)` Breite des Quellrasters
`bb%(1)` Höhe des Quellrasters

In den beiden ersten Feldelementen sind die Breite und die Höhe des Quellrasters zu finden.

`bb%(2)` Anzahl der Farbebenen
`bb%(3)` Vordergrundfarbe
`bb%(4)` Hintergrundfarbe

Bei Verwendung eines Schwarzweiß-Monitors gibt es nur eine Farbebene. Als Farbwerte können dann nur die Werte 0 (weiß) oder 1 (schwarz) angegeben werden.

`bb%(5)` Verknüpfungsmodus des Ziels mit der Quelle

Sie haben die Wahl zwischen 16 Verknüpfungsmodi. Jeder Modus muß in Langwortbreite, übergeben werden. Beispiel: Modus &H3030303 überträgt das Quellraster 1:1 in das Zielraster. Die folgende Tabelle gibt Ihnen Aufschluß über die möglichen Modi, jeder Modus wird in hexadezimaler Schreibweise angegeben:

Hex	bewirkt...	Bemerkung
00	Ziel = löschen	
01	Ziel = Quelle AND Ziel	
02	Ziel = Quelle AND (NOT Ziel)	
03	Ziel = Quelle	Übertragung 1:1
04	Ziel = (NOT Ziel) AND Quelle	Löscht Ziel
05	Ziel = Ziel	Ziel bleibt unverändert
06	Ziel = Quelle XOR Ziel	XOR-Modus
07	Ziel = Quelle OR Ziel	
08	Ziel = NOT (Quelle OR Ziel)	
09	Ziel = NOT (Quelle XOR Ziel)	
0A	Ziel = NOT Ziel	
0B	Ziel = Quelle OR (NOT Ziel)	
0C	Ziel = NOT Quelle	
0D	Ziel = (NOT Ziel) OR Quelle	
0E	Ziel = NOT (Quelle AND Ziel)	
0F	Ziel = 1	Alle Zielpunkte setzen

Das waren die Modi. Wem die Bemerkungen zu spärlich erscheinen, der schlage bitte im Grafikkapitel nach. Dort befindet sich auch ein Demoprogramm, das die Modi anschaulich auf dem Bildschirm darstellt. Es folgt die Belegung der übrigen Feldelemente.

`bb%(6)`	X-Koordinate der Quelle
`bb%(7)`	Y-Koordinate der Quelle
`bb%(8)`	Adresse des Quellrasters

Die nächsten Parameter enthalten die Koordinaten der linken oberen Ecke und die Speicheradresse des Quellrasters.

`bb%(9)`	Offset zum nächsten Quell-Wort in Byte
`bb%(10)`	Offset zur nächsten Quell-Zeile in Byte
`bb%(11)`	Offset zur nächsten Quellen-Farb-Ebene (immer 2)

Die obenstehenden Parameter beschreiben die Quelle näher. Das nächste Wort einer Zeile wird wohl immer zwei Byte weiter beginnen.

Hinweis: Eine neue Zeile muß immer an einer Wortgrenze beginnen. Übergabe an »bb%(10)« von Werten wie 1, 3 usw. sind nicht erlaubt.

`bb%(12)` X-Koordinate des Ziels
`bb%(13)` Y-Koordinate des Ziels
`bb%(14)` Zieladresse
`bb%(15)` Offset zum nächsten Ziel-Wort in Byte
`bb%(16)` Offset zur nächsten Ziel-Zeile in Byte (siehe Hinweis)
`bb%(17)` Offset zur nächsten Ziel-Farb-Ebene (immer 2)

Diese Einträge beschreiben das Zielraster näher. Ihre Bedeutung entnehmen Sie bitte den entsprechenden Beschreibungen des Quellrasters.

Das Quellraster kann beim Transport mit einem selbstdefinierten Füllmuster logisch UND-verknüpft werden. Erst nach der Verknüpfung erfolgt dann die in »bb%(5)« angegebene Verknüpfung mit dem Zielbereich. Das Füllmuster muß von Ihnen in einem beliebigen Speicherbereich angelegt werden. Auch bei ihm beginnt eine neue Zeile immer an einer Wortgrenze. Die Maße des Musters beschreiben folgende Einträge des Arrays:

`bb%(18)` Adresse des Füllmusters
`bb%(19)` Offset zur nächsten Zeile des Musters in Byte
`bb%(20)` Offset zur nächsten Farb-Ebene des Musters in Byte
`bb%(21)` Höhe des Füllmusters

Soviel zum Transport der Bildschirmdaten in den Speicher und zurück. Wir kommen zum Listing.

```
GOSUB definitionen
GOSUB zeichen_bildschirm_einrichten
GOSUB bildschirm_loeschen
GOSUB menue_leiste_erstellen
GOSUB auf_ereignisse_warten
```

Das Programm besitzt einen modularen Aufbau. Somit ist es für Sie später recht leicht, eine eigene Fensterverwaltung aufzubauen. Bedingt durch den Aufbau, werden die einzelnen Programmteile mit »GOSUB« aufgerufen. Die ersten vier Unterprogrammaufrufe dienen der Festlegung der Arbeitsumgebung.

```
'
> PROCEDURE definitionen
  ON ERROR GOSUB ende
  ON MENU GOSUB menue_auswertung
  ON MENU MESSAGE GOSUB auswertung
  ON MENU BUTTON 1,1,1 GOSUB zeichnen
```

Die Abfrage der Benutzeraktivitäten wird ausschließlich über »ON MENU«-Aufrufe realisiert.

```
'
RESERVE 1024                          !1024 Byte fürs Programm
fr_sp%=MALLOC(-1)                     !Freien Speicher ermitteln
SUB fr_sp%,20000                      !Etwas fürs Betriebssystem
speicher_ad%=MALLOC(fr_sp%)           !Den Rest für den Bildschirm
```

Die nächsten Zeilen dienen der Speicherorganisation. Zuerst gibt das Programm, bis auf 1024 Byte für die Variablen, den noch verfügbaren Speicher an das GEM-DOS zurück. Durch den ersten MALLOC-Aufruf erfahren wir die Größe des Speichers. Da das GEM selbst auch noch Speicher für die interne Verwaltung benötigt, reservieren wir lediglich die erhaltene Größe minus 20.000 Byte. Die Anfangsadresse steht nach dem zweiten MALLOC-Aufruf in der Variablen »speicher_ad%«.

```
DIM t_ad%(7)                          !Adresse für Titel-String
DIM i_ad%(7)                          !Adresse für Infozeilen-String
FOR i&=0 TO 6                         !Adressen festlegen
t_ad%(i&)=speicher_ad%+i&*80
i_ad%(i&)=speicher_ad%+560+i&*80
NEXT i&
```

Jedes der möglichen sieben Fenster soll später über eine Titel- und eine Infozeile verfügen. Für die Texte werden jeweils 80 Byte reserviert. Die Adressen der Texte sind später in den jeweiligen Array-Einträgen zu finden.

```
DIM m_eintrag$(30)                    !Feld für Menüeinträge
```

Das Programm verfügt auch über eine Menüleiste, die vom Basic angelegt wird. In »m_eintrag$()« stehen später die Titel und Einträge der Leiste.

```
DIM bz&(7)                            !Feld für Bildschirmzeilen
DIM bs&(7)                            !Feld für Bildschirmspalten
```

Für jedes Fenster kann später ein individuell wählbarer Bereich des Grafikbildschirms angezeigt werden. Das Programm merkt sich in den Feldern »bz&()« und »bs&()« die jeweiligen Koordinaten der linken oberen Ecke des angezeigten Bereichs.

```
DIM x_a&(7)                           !Feld für Voll-Feld-Maße
DIM y_a&(7)                           !Feld für Voll-Feld-Maße
DIM b_a&(7)                           !Feld für Voll-Feld-Maße
DIM h_a&(7)                           !Feld für Voll-Feld-Maße
DIM bb%(27)                           !Feld für BITBLT (LINE A)
```

Vergrößert der Anwender ein Fenster über das Voll-Feld, rettet das Programm die aktuelle Größe in die Arrays »x_a&()« bis »h_a&()«. Damit ist sichergestellt, daß, nach nochmaligem Anklicken des Voll-Feldes, die alte Größe wiederhergestellt werden kann. Die Funktion des Feldes »bb%()« haben wir Ihnen bereits oben vorgestellt.

```
DEFFILL 1,2,4    !Füllmuster für Desktop
```

Obiger Befehl bestimmt das Füllmuster für unser Desktop.

```
'
maxx&=0                                !Kleinste X-Koordinaten
maxy&=0                                !Kleinste Y-Koordinaten
maxb&=1000                             !Maximale Bild-Breite
maxh&=1000                             !Maximale Bild-Höhe
```

Vorstehende Variablen beschreiben den Grafikbildschirm. Je nachdem, wieviel Speicherplatz Ihnen zu Verfügung steht, müssen Sie die Breite und Höhe verkleinern oder können Sie sogar noch innerhalb erwähnter Grenzen vergrößern.

```
'
' ******** FELD FÜR BITBLT VORBELEGEN ***********
'
bb%(2)=1                               !Anzahl der Farb-Ebenen
bb%(3)=1                               !Vordergrundfarbe
bb%(4)=0                               !Hintergrundfarbe
bb%(9)=2                               !Offset zum nächsten Wort (Von) in Byte
bb%(11)=2                              !Offset zur nächsten !Farb-Ebene (Quelle)
bb%(15)=2                              !Offset zum nächsten Wort (Ziel)
bb%(17)=2                              !Offset zur nächsten Farb-Ebene (Ziel)
bb%(18)=0                              !Zeiger auf Tabelle mit Füllmuster
bb%(19)=0                              !Offset zur nächsten Zeile !der Füllmaske
bb%(20)=0                              !Offset zur nächsten Farbe !der Füllmaske
bb%(21)=0                              !Höhe der Füllmaske
RETURN
```

Am Ende des Unterprogramms werden die Teile des BITBLT-Arrays vorbelegt, die sich nicht ändern.

```
> PROCEDURE zeichen_bildschirm_einrichten
  ad_bs%=speicher_ad%+1120             !Adresse des Zeichenbs.
  akt_bs%=XBIOS(&H2)                   !Adresse des aktuellen Bs.
RETURN
```

Die Routine erledigt das Berechnen der Anfangsadresse des Grafikbildschirms. Sie liegt, wegen der Titel- und Infozeilen, 1120 Byte höher als zurückerhalten und stellt dann die Anfangsadresse des aktuellen Bildschirms fest.

```
> PROCEDURE bildschirm_loeschen
  bb%(0)=maxb&                         !Breite des Quell-Rasters
  bb%(1)=maxh&                         !Höhe des Quell-Rasters
  bb%(5)=&H0                           !Verknüpfungsmodus (Ziel löschen)
  bb%(6)=0                             !X-Koordinate Quelle
```

```
  bb%(7)=0                                !Y-Koordinate Quelle
  bb%(8)=ad_bs%                           !Adresse der Quelle
  bb%(10)=(((maxb&/16) AND 65534)+2)*2    !Offset nächste
                                          !Q.-Zeile in Byte
  bb%(12)=0                               !X-Koordinate Ziel
  bb%(13)=0                               !Y-Koordinate Ziel
  bb%(14)=ad_bs%                          !Ziel-Adresse
  bb%(16)=(((maxb&/16) AND 65534)+2)*2    !Offset nächste Z.-Zeile
                                          !in Byte
  BITBLT bb%()
RETURN
```

Der Grafikbildschirm wird mit Hilfe der vorgestellten BITBLT-Routine gelöscht. Der Befehl »CLS« würde, auch nach Umschalten auf den Grafikbildschirm, kein befriedigendes Ergebnis bringen. Er löscht nämlich nur 32.000 Byte, zu wenig für unsere Anwendung. Die Berechnung der Variablen »bb%(10)« und »bb%(16)« stellt sicher, daß die nächste Zeile des Grafikbildschirms auch wirklich an einer Wortgrenze liegt. Die Dauer des Löschvorgangs ist sehr stark von der gewählten Bildschirmauflösung abhängig. Ein eingebauter und aktivierter Blitter beschleunigt den Vorgang etwa um den Faktor 10.

Das folgende Unterprogramm baut die Menüleiste auf und stellt sie dar. Danach wird das Desktop eingerichtet.

```
> PROCEDURE menue_leiste_erstellen
  LOCAL i&
  DO                                      !Menüeinträge einlesen
 READ m_eintrag$(i&)
 EXIT IF m_eintrag$(i&)="-1"
 INC i&
  LOOP
  '
  MENU m_eintrag$()                       !Menüleiste erstellen lassen
```

Den obigen Vorgang, das Anlegen der Menüleiste, haben wir Ihnen schon im Kapitel 3 erklärt, wir brauchen nicht mehr darauf einzugehen. Bei Bedarf lesen Sie bitte noch einmal nach.

```
  CLIP 0,18,640,399                       !Übrigen Bildschirm einrichten
  PBOX 0,18,639,399
```

Sind Sie ob obiger Befehle verwundert? Nun, wir hatten zunächst auch den Befehl »OPENW(0)« eingesetzt. Beim Experimentieren mit dem Programm stellten wir dann aber fest, daß die Fensterfunktionen des GEM den Bildschirmursprung in der linken oberen Ecke des Bildschirms erwarten. GFA setzt den Ursprung nach »OPENW(0)« jedoch um 19 Zeilen tiefer. Dieser Umstand erzeugte einige ungewollte Effekte, darum sorgen wir durch den CLIP-Befehl dafür, daß die Menüleiste nicht übermalt werden kann.

```
'
  SELECT XBIOS(64,-1) AND &X11          !Blitter vorh. und eingesschaltet
  CASE 0                                !Kein Blitter vorhanden
 MENU 16,2
  CASE 3                                !Blitter vorh. und eingeschaltet
 MENU 16,1
  ENDSELECT
```

Ein entsprechender Menüeintrag läßt das Ein- oder Ausschalten eines evtl. vorhandenen Blitters zu. Stolzen Besitzern wird so recht anschaulich der Geschwindigkeitsvorteil demonstriert. Den Rest des Unterprogramms bilden die DATA-Zeilen mit den Menüeinträgen.

```
'
  DATA DESK, NICHTS ,--------------------,1,2,3,4,5,6,""
  DATA  FENSTER , ÖFFNEN , INHALT LÖSCHEN , ALLE NEU ZEICHNEN ,""
  DATA  SONSTIGES ,  BLITTER  ,  SCHLUSS  ,""
  DATA "",""
  DATA -1
RETURN
```

Die Vorbereitungen sind beendet. Jetzt heißt es für das Programm, auf die Anforderungen des Anwenders zu reagieren: Eine nicht immer leichte Angelegenheit. Das Warten übernimmt die untenstehende Prozedur.

```
> PROCEDURE auf_ereignisse_warten
  DO
 ON MENU
  LOOP
RETURN
```

Solange kein Fenster auf dem Desktop liegt, können sich die Anforderungen nur auf die Menüleiste beziehen. Die Aktionen wertet die untenstehende Routine aus.

```
> PROCEDURE menue_auswertung
  LOCAL e$
  e$=m_eintrag$(MENU(0))
  IF e$=" ÖFFNEN "
 a_ha%=ha%                              !Altes Handle retten
 GOSUB fenster_anlegen(4095," Fenster ","",0,0,639,399,ha%)
 IF ha%>0 AND ha% 8                     !Anlegen hat geklappt
   fm&=BSET(fm&,ha%)                    !Bit der Fenstermaske setzen
   GOSUB fenster_oeffnen(ha%)
   ~WIND_GET(ha%,4,x&,y&,b&,h&)         !Größe Arbeitsbereich
   GOSUB vertikalen_slider_setzen(x&,y&,b&,h&)
   GOSUB horizontalen_slider_setzen(x&,y&,b&,h&)
```

```
ELSE                                    !Anlegen hat nicht geklappt
  ha%=a_ha%                             !Gerettetes Handle zurück
ENDIF
```

Der String des angeklickten Menüeintrags wird in die Variable »e$« kopiert. Hat der Anwender ÖFFNEN gewählt, rettet das Programm zunächst das alte Handle und verzweigt zum Unterprogramm »fenster_anlegen«. Das Unterprogramm liefert uns das Handle zurück. Bei einem Handle kleiner als 8, war noch ein Fenster zu vergeben. In der Fenstermaske »fm&« wird das entsprechende Bit gesetzt. Zum späteren Schließen aller Fenster ist es nämlich wichtig, zu wissen, welche Fenster geöffnet sind. Anschließend kann das Fenster geöffnet werden. Als nächstes erfragt das Programm die Größe des Arbeitsbereichs und verzweigt in zwei weitere Prozeduren, die den vertikalen und horizontalen Slider und deren Größe setzen. Um den Fensterinhalt brauchen wir uns nicht zu kümmern, da das AES uns nach dem Öffnen eine REDRAW-Message zukommen läßt. Der Inhalt wird dadurch über die entsprechende Routine erstellt. War kein Handle mehr frei, schreibt das Programm das alte wieder zurück in die Variable »ha%«.

```
 ELSE IF e$=" INHALT LÖSCHEN "
GOSUB bildschirm_loeschen
GOSUB alles_neu_zeichnen
 ELSE IF e$=" ALLE NEU ZEICHNEN "
GOSUB alles_neu_zeichnen
 ELSE IF e$="  BLITTER  "
SELECT XBIOS(64,-1) AND 1               !Blitter eingeschaltet?
CASE 1          !Ja
  MENU MENU(0),0                        !Checkmark weg
  ~XBIOS(64,0)                          !Blitter ausschalten
CASE 0                                  !Nein
  MENU MENU(0),1                        !Checkmark
  ~XBIOS(64,1)                          !Blitter einschalten
ENDSELECT
 ELSE IF e$="  SCHLUSS  "
GOSUB ende
 ENDIF
 MENU OFF
RETURN
```

Obenstehenden Teil brauchen wir wohl nicht mehr zu erklären, er erklärt sich über die REM-Zeilen und Unterprogrammnamen von selbst. Die aufgerufenen Prozeduren erläutern wir nach und nach, den Anfang machen wir mit dem Anlegen und Öffnen eines Fensters.

Der Prozedur »fenster_anlegen« übergeben Sie die gewünschten Attribute, den Namen, den Eintrag der Info-Zeile und die Koordinaten und die maximal mögliche Größe des Fensters. Zurück gibt sie das Handle oder, wenn der Aufruf keinen Erfolg hatte, eine Fehlernummer des GEM (65535). Wir haben auf eine Überprüfung der gewünschten Koordinaten und

Größe mit den wirklich möglichen verzichtet, da sich unser Demo an die Grenzen des Erlaubten eines monochromen Monitors hält. Möchten Sie ein vollkommen portables Programm unter Verwendung dieses Unterprogramms schreiben, so sollten Sie auf eine Überprüfung des gewünschten mit dem erlaubten Bereich nicht verzichten.

```
> PROCEDURE fenster_anlegen(att&,name$,info$,x&,y&,b&,h&,VAR ha%)
  LOCAL x&,y&,b&,h&,mx&,my&,mb&,mh&,in&
  ~WIND_GET(0,4,mx&,my&,mb&,mh&)          !Desktopgröße ermitteln
  ~WIND_CALC(1,att&,mx&,my&,mb&,mh&,x&,y&,b&,h&)
  ~WIND_CALC(0,att&,x&,y&,b&,h&,mx&,my&,mb&,mh&)
  ha%=WIND_CREATE(att&,mx&,my&,mb&,mh&) !Fenster anlegen
  '
  IF ha%>0 AND ha% 8                      !Wenn Anlegen erfolgreich
 in&=ha%-1
 IF BTST(att&,0)                          !Wenn Namenszeile erwünscht
   CHAR{t_ad%(in&)}=LEFT$(name$+STR$(ha%),80)
   ~WIND_SET(ha%,2,INT(t_ad%(in&)/2^16),MOD(t_ad%(in&),2^16),0,0)
 ENDIF
 IF BTST(att&,4)                          !Wenn Infozeile erwünscht
   CHAR{i_ad%(in&)}=LEFT$(info$,80)
   ~WIND_SET(ha%,3,INT(i_ad%(in&)/2^16),MOD(i_ad%(in&),2^16),0,0)
 ENDIF
  ENDIF
RETURN
```

Den Ablauf des Unterprogramms haben wir schon bei den allgemeinen Erklärungen erläutert. Das Programm erfragt zuerst die Größe des Desktops und danach die Gesamt- und Arbeitsbereichsgröße eines gedachten Fensters mit maximalen Maßen. War der Aufruf erfolgreich (es war noch ein Handle frei) und sind Titel und/oder Info-Zeile erwünscht, legt sie das Programm, über entsprechende WIND_SET-Aufrufe, ebenfalls an. Die nächste Prozedur öffnet das eben angelegte Fenster.

```
> PROCEDURE fenster_oeffnen(ha%)
  LOCAL x&,y&,b&,h&
  x&=RANDOM(220)+20                      !Zufällige Koordinaten erzeugen
  y&=RANDOM(300)+20                      !Zufällige Koordinaten erzeugen
  b&=RANDOM(200)+200                     !Zufällige Koordinaten erzeugen
  h&=RANDOM(200)+150                     !Zufällige Koordinaten erzeugen
  ~GRAF_GROWBOX(0,0,0,0,x&,y&,b&,h&)
  ~WIND_OPEN(ha%,x&,y&,b&,h&)   !FENSTER ÖFFNEN
RETURN
```

Damit die Fenster nicht alle übereinander liegen, erzeugt das Programm innerhalb bestimmter Grenzen zufällige Koordinaten und Fenstergrößen. Einen lediglich optischen Effekt erzeugt der Aufruf der Funktion »GRAF_GROWBOX«. Er zeichnet ein größer wer-

dendes Rechteck mit den Anfangskoordinaten und der Größe (0,0,0,0) und den Endkoordinaten und der Endgröße (x&,y&,b&,h&). Der Befehl »WIND_OPEN« läßt das Fenster schließlich auf dem Desktop erscheinen.

So, wie es kommt, daß ein Fenster erscheint, wissen Sie jetzt. Eine Auswertroutine ist ausschließlich für die Fensterverwaltung zuständig. Sie wertet alle eintreffenden Nachrichten des AES aus. Die untenstehende Routine wird also nur angesprungen, wenn mindestens ein Fenster geöffnet ist und mit ihm irgendwelche Dinge geschehen sollen.

```
> PROCEDURE auswertung
  a_ha%=ha%                          !Altes Handle retten
  ha%=MENU(4)                        !Welches Fenster ist gemeint?
  ~WIND_UPDATE(1)                    !Alle Aktivitäten stop!
```

In der Variablen »ha%« befindet sich immer das Handle des aktuellen Fensters. Einige GEM-Funktionen geben, z.B. beim Neuzeichnen, das Handle eines anderen, ebenfalls sichtbaren, aber nicht aktiven Fensters zurück. Das Programm rettet daher das aktuelle Handle in die Variable »a_ha%«. Das Handle (die Fensternummer), das sich auf das Fenster bezieht, ist immer im Eintrag »MENU(4)« zu finden. Der Variablen »ha%« wird deshalb die Nummer übergeben. Die nächste Zeile bedarf einiger Erläuterungen.

Es gibt Aktionen, die es erforderlich machen, dem Anwender die Kontrolle über das Programm für einen gewissen Zeitraum zu entziehen. Dazu zählt z.B. das Neuzeichnen von Fensterteilen. Es kann sonst nämlich passieren, daß das Programm schon wieder neue Aktivitäten auswertet, ohne vollständig auf vorherige entsprechend reagiert zu haben. Das Chaos ist perfekt. Den Stopp bzw. den erneuten Beginn der Abfrage veranlaßt der Befehl:

~WIND_UPDATE(P&)

»P&« bestimmt, auf welche Aktivitäten das GEM nicht mehr bzw. wieder reagieren soll.

P&	Wirkung
0	Bildschirmaufbau ist beendet, neue Aktivitäten werden wieder zugelassen.
1	Bildschirmaufbau beginnt, alle Aktivitäten, die die Fenster betreffen, werden unterbunden.
2	Das Programm übergibt die Kontrolle der Maus wieder an das GEM (das ist der Regelzustand).
3	Die Mauskontrolle wird an das Programm übergeben. Dabei ist die Abfrage der Menüleiste nur noch über dieses (durch Positionsabfrage) möglich.

Unser Programm läßt keine weiteren Fensteraktionen mehr zu. Der nächste Unterprogrammteil analysiert die eingetroffene Nachricht weiter und entscheidet, zu welchem anderen Unterprogramm verzweigt werden muß.

```
  SELECT MENU(1)
```

Wie im Eingangsteil bereits erwähnt, steht die Nachrichtennummer immer im ersten Wort des Message-Buffers.

```
  CASE 20                            !Redraw-Message
 GOSUB neu_zeichnen
  CASE 21                            !Anderes Fenster soll aktiv werden
 GOSUB fenster_top
  CASE 22
 GOSUB fenster_schliessen            !Closer angeklickt
  CASE 23
 GOSUB voll_feld                     !Voll-Feld angeklickt
  CASE 24
 GOSUB scrollen_1                    !Es soll gescrollt werden
  CASE 25,26
 GOSUB scrollen_2                    !Ein Slider wurde verschoben
  CASE 27
 GOSUB fenster_groesse               !Fenstergröße wurde verändert
  CASE 28
 GOSUB fenster_bewegung              !Fenster wurde verschoben
  ENDSELECT
  ~WIND_UPDATE(0)                    !Aktivitäten wieder zulassen
RETURN
```

Wichtig ist in diesem Unterprogramm, daß mit »WIND_UPDAT(0)« die Fensteraktivitäten wieder zugelassen werden. Die jeweiligen Unterprogramme erklären wir als nächstes. Bevor wir den Ablauf und die Bedeutung der nächsten Prozedur erklären, weitere Grundlagen.

4.3.1 Neuzeichnen von Bildschirmteilen

Bevor Sie weiterlesen, legen Sie sich bitte einige Blätter Papier, einen Stift und eine Schere zurecht. Diese Teile dienen keinem Hardware-Eingriff, sondern werden Ihnen das Verstehen der folgenden Zusammenhänge bestimmt erleichtern.

Ein Blatt bleibt in der Originalgröße bestehen, es soll das Desktop darstellen. Kennzeichnen Sie es bitte entsprechend. Jetzt schneiden Sie sich drei »Fenster« (Rechtecke) zurecht. Jedes Fenster muß kleiner als das »Desktop« sein und soll ein anderes Muster erhalten. Positionieren Sie dann die vorbereiteten Rechtecke so auf dem »Desktop«, daß sie sich teilweise überlappen. Verschieben Sie das oberste »Fenster« in eine beliebige Richtung. Sie sehen, daß Teile der anderen »Fenster«, die vorher verdeckt waren, sichtbar

werden. Fast genauso ist es bei unserem Rechner, nur noch etwas komplizierter. Während bei unserem »Papierversuch« die verdeckten Informationen nach dem Verschieben sofort wieder sichtbar wurden, muß man sie beim Computer erst wieder mühevoll aufbauen. Räumen Sie die Fenster bitte wieder vom »Desktop«, es folgt die zweite Demonstration.

Als zweiten Versuch legen Sie bitte wieder ein Fenster auf das »Desktop«. Darüber plazieren Sie ein weiteres so, daß es einen Teil des ersten überdeckt. Jetzt markieren Sie mit dem Bleistift die Umrisse des zweiten Fensters und nehmen es danach weg. Übrig bleiben das erste Fenster und die Umrisse des zweiten. Uns als Programmierer interessieren lediglich die Bereiche des ersten Fensters innerhalb der Markierung. Das Desktop wird nämlich vom GEM selbst auf dem laufenden gehalten, es gibt trotzdem genug zu tun.

Nehmen wir an, wir hätten vor dem Positionieren des zweiten Fensters im *Überlappungsbereich* die Informationen des ersten wegradiert. In diesem Fall wäre nach dem Entfernen ein weißes Feld im ersten Fenster. Die Informationen, die vorher an dieser Stelle waren, sind verloren. Wir haben aber vorgesorgt und vor dem Öffnen des zweiten Fensters eine genaue Kopie des Überlappungsbereichs angefertigt. Sie wurde an sicherer Stelle verwahrt. Jetzt, nachdem das zweite Fenster nicht mehr vorhanden ist, übertragen wir die Kopie in den weißen Bereich, und alles ist in Ordnung. Das war die einfachste Möglichkeit.

Wir machen die Sache noch komplizierter. Dazu wird das zweite Fenster nicht entfernt, sondern so verschoben, daß nur ein kleiner Teil des überdeckten Bereichs wieder sichtbar wird. Wenn wir jetzt die Kopie nehmen und alles übertragen, dann..., ja dann wäre der Inhalt des zweiten Fensters zerstört. Wir glauben, daß Sie die Problematik erkannt haben, ganz »verrückt« wird es, wenn man mit noch mehr Fenstern arbeitet.

GEM ist zum Glück nicht so konzipiert, daß das Restaurieren des Bildschirms einzig und allein dem Programmierer überlassen bleibt. Es stellt ihm einige Hilfsmittel zur Verfügung:

- Führt der Benutzer des Programms einen Schritt aus, der das Neuzeichnen eines bestimmten Bildschirmbereichs erforderlich macht, sendet das AES eine Nachricht an das Programm. Diese Nachricht enthält das Handle des Fensters und die Koordinaten des neu zu zeichnenden Bereichs. Müssen zum Beispiel Teile von zwei Fenstern restauriert werden, so werden auch zwei Nachrichten abgesetzt.
- Das Programm braucht sich nicht um das Restaurieren der Randbereiche, also Titelzeile, Slider usw., zu kümmern, diese Aufgabe nimmt ihm das AES ab.
- AES verwaltet für jedes Fenster eine Rechteckliste. Sie besteht zum Beispiel für das aktive Fenster aus nur einem Eintrag. Er beschreibt die Koordinate der linken oberen Ecke und die Breite und Höhe des Arbeitsbereichs. Es können aber auch mehrere Einträge vorhanden sein, denn AES »merkt« sich jedes Rechteck, das überdeckt wird. Die Einsicht in die Rechteckliste kann jederzeit vom Programm angefordert werden (es gibt keinen Datenschutz).

Dazu ein Beispiel; alle Koordinatenangaben beziehen sich auf den Arbeitsbereich:

Die linke obere Ecke des Fensters 1 ist bei 100,100, seine Breite betrage 100 und seine Höhe ebenfalls 100 Pixel. Auf diesem Fenster befindet sich Fenster 2. Seine Koordinaten: 150,150, die Breite und Höhe betragen jeweils 200 Pixel. Es überdeckt also das erste Fenster im Bereich 150,150 mit einer Breite und Höhe von jeweils 50 Pixel. Diese Werte werden vom AES automatisch in die Rechteckliste eingetragen. Der Anwender verschiebt Fenster 2 auf die Position 175,175, Breite und Höhe bleiben beim Verschieben ja gleich. Fenster 2 überdeckt jetzt noch den Bereich 175,175, Breite und Höhe betragen 25 Pixel. AES sendet jetzt an das Programm eine Nachricht. Übersetzt sieht sie so aus:

- Der Bereich des Fensters 1 muß neu gezeichnet werden. Die linke obere Ecke dieses Bereichs liegt bei 150,150, seine Breite beträgt 25 Pixel und seine Höhe ebenfalls 25 Pixel.

Das Programm muß auf diese Nachricht folgendermaßen reagieren:

- Alle Fensteraktionen müssen gestoppt werden.
- Es fordert den ersten Eintrag der Rechteckliste des Fensters 1 an.
- AES übergibt dem Programm, wie schon erwähnt, die Koordinaten 150,150 mit einer Breite von 50 und einer Höhe von 50 Pixel. Restauriert nun das Programm diesen Bereich, so ist leicht zu erkennen, daß dabei Teile des Fensters 2 zerstört werden. Daher müssen noch die Schnittpunkte zwischen dem neu zu zeichnenden Bereich und den Angaben aus der Rechteckliste berechnet werden. Ist das Ergebnis positiv (gemeint ist das Vorzeichen), restauriert das Programm nur diesen Bereich.
- Jetzt werden die übrigen Einträge der Rechteckliste abgefordert. In unserem Beispiel sind keine mehr vorhanden. AES signalisiert das dadurch, daß es für Breite und Höhe 0 oder negative Werte übergibt. Das Neuzeichnen des Fensters ist damit beendet.

Wir bieten Ihnen die folgende Lösung dieses vielfältigen Problems an:

```
'
> PROCEDURE neu_zeichnen
  LOCAL x&,y&,b&,h&,x1&,y1&,b1&,h1&
  ~WIND_GET(ha%,4,x1&,y1&,b1&,h1&)       !Größe des neu zu
                                         !zeichnenden Fensters
  ~WIND_UPDATE(1)                        !Alle Aktivitäten stop
```

Zuerst wird die Arbeitsbereichsgröße des neu zu zeichnenden Fensters erfragt. Danach stoppt das Unterprogramm alle weiteren Fensteraktivitäten. Das ist in diesem Programm eigentlich überflüssig, da bereits geschehen. Das Unterprogramm sollte aber universell einsetzbar sein.

```
  ~WIND_GET(ha%,11,x&,y&,b&,h&)          !Erstes Rechteck anfordern
```

Diese Funktion fordert das erste Rechteck der Rechteckliste für das Fenster mit dem Handle »ha%« an. Die folgende Schleife wird nur so oft durchlaufen, wie Werte für »b&« und »h&« größer 0 von »WIND_GET(.,11,.,.,.,.)« oder »WIND_GET(.,12,.,.,.,.)« zurückgeliefert werden.

```
  WHILE b&>0 AND h&>0                       !Ende wenn Höhe oder
          !Breite  =0
  ~RC_INTERSECT(MENU(5),MENU(6),MENU(7),MENU(8),x&,y&,b&,h&)
```

Die Einträge des Message-Buffers MENU(5) bis MENU(8) enthalten die Koordinaten der neu zu zeichnenden Fläche. Die GFA-Funktion »RC_INTERSECT« berechnet die Koordinaten eines evtl. vorhandenen Schnittpunktes. Sie stehen nach dem Aufruf in den Variablen »x&« bis »h&«.

```
 IF b&>0 AND h&>0                           !Schnittfläche vorhanden
   sp&=x&-x1&+bs&(ha%)                      !Spalte für Neuzeichnen berechnen
   ze&=y&-y1&+bz&(ha%)                      !Zeile für Neuzeichnen berechnen
   GOSUB inhalt_restaurieren(x&,y&,b&,h&,sp&,ze&)
 ENDIF
 ~WIND_GET(MENU(4),12,x&,y&,b&,h&)          !Weiteres Rechteck anfordern
   WEND
  ~WIND_UPDATE(0)                           !Aktivitäten wieder zulassen
RETURN
```

Ist ein Schnittpunkt vorhanden, berechnet das Programm die Spalte und Zeile des neu zu zeichnenden Bereichs. Danach verzweigt es in ein Unterprogramm, das Daten aus dem Grafikbildschirm auf den Bildschirm transportiert. Sind alle Teile auf dem neuesten Stand, läßt das Programm die gesperrten Aktivitäten wieder zu (Bemerkung siehe oben). Der Programmteil, der das Neuzeichnen realisiert, soll Ihnen nicht länger vorenthalten bleiben. Er sieht so aus:

```
> PROCEDURE inhalt_restaurieren(x&,y&,b&,h&,sp&,ze&)
  HIDEM                                     !Maus aus
  bb%(0)=b&                                 !Breite des Rasters der Quelle
  bb%(1)=h&                                 !Höhe des Rasters der Quelle
  bb%(5)=&H3030303                          !Verknüpfungsmodus
  bb%(6)=sp&                                !X-Koordinate Quelle
  bb%(7)=ze&                                !Y-Koordinate Quelle
  bb%(8)=ad_bs%                             !Adresse der Quelle
  bb%(10)=(((maxb&/16) AND 65534)+2)*2      !Offset nächste Quellzeile in Byte
  bb%(12)=x&                                !X-Koordinate Ziel
  bb%(13)=y&                                !Y-Koordinate  Ziel
  bb%(14)=akt_bs%                           !Adresse des Ziels
  bb%(16)=80                                !Offset nächste Ziel-Zeile in Byte
  BITBLT bb%()
```

```
    i$="Sp= "+STR$(bs&(ha%))+" Z= "+STR$(bz&(ha%))
    i$=i$+" B= "+STR$(b&)+" H= "+STR$(h&)
    CHAR{i_ad%(ha%-1)}=i$
    ~WIND_SET(ha%,3,INT(i_ad%(ha%-1)/2^16),MOD(i_ad% (ha%-1),2^16),0,0)
    SHOWM                               !Maus wieder ein
RETURN
```

Zuerst schaltet das Programm den Mauscursor aus, es bleiben sonst nämlich evtl. häßliche Flecken zurück, und das wollen wir ja nicht. In der Variablen »bb%(5)« steht der Verknüpfungsmodus in Langwortbreite, in unserem Fall: Zielinhalt soll Quellinhalt sein, d.h., es erfolgt eine 1:1-Übertragung. Die Berechnung von »bb%(10)« ergibt, daß jede Zeile an einer Wortgrenze beginnt, Sie kennen das ja schon. Nachdem der entsprechende Grafikausschnitt in das Fenster übertragen wurde, wird die Info-Zeile auf den aktuellen Stand gebracht. Das Unterprogramm führt diesen Schritt immer aus, ganz egal, ob notwendig oder nicht. Sie können das ja verbessern. Den Schluß bildet das Einschalten des Mauscursors.

Es gibt Situationen, die es erforderlich machen, den gesamten Bildschirmaufbau neu zu zeichnen. Unser Programm verfügt zum Beispiel über einen entsprechenden Menüpunkt. Aber auch wenn Sie den Grafikspeicher löschen, werden die Inhalte aller auf dem Desktop vorhandenen Fenster restauriert, sprich gelöscht. Dieser Vorgang wird nicht vom AES eingeleitet. Woher sollte es auch wissen, daß das Programm gerade den Grafikspeicher gelöscht hat. Auslöser ist das folgende Unterprogramm.

```
> PROCEDURE alles_neu_zeichnen
    LOCAL x&,y&,b&,h&
    ~WIND_GET(0,4,x&,y&,b&,h&)                 !Größe des Desktops
    ~FORM_DIAL(0,x&,y&,b&,h&,x&,y&,b&,h&) !Redraw-Message erzwingen
    ~FORM_DIAL(3,x&,y&,b&,h&,x&,y&,b&,h&) !Redraw-Message erzwingen
RETURN
```

Zuerst besorgt sich das Programm die Größe des Desktops. Mit dem ersten FORM_DIAL-Aufruf gaukeln wir GEM vor, daß wir beabsichtigen, eine Dialogbox der erhaltenen Größe auf den Bildschirm zu bringen. Der zweite FORM_DIAL-Aufruf »schließt« die nie geöffnete Box wieder. Das war's schon. AES vergleicht jetzt nämlich die Koordinaten, und stellt fest, daß alle Fenster unter der imaginären Dialogbox lagen, und sendet daher eine entsprechende Redraw-Message an das Programm. Die Reaktion darauf kennen Sie schon. Soviel zum Neuzeichnen, die anderen Benutzeranforderungen sind nicht ganz so kompliziert.

4.3.2 Die anderen Nachrichten

Den meisten der jetzt folgenden Nachrichten sendet das AES noch eine weitere hinterher. Es ist die eben vorgestellte, die Redraw-Message. Der Programmierer braucht sich deshalb nach ihrer Programmierung fast nie mehr Gedanken über den Fensterinhalt zu machen.

```
> PROCEDURE fenster_top
  ~WIND_SET(ha%,10,n&,n&,n&,n&)             !Angeklicktes Fenster setzen
RETURN
```

Vorstehende Prozedur springt das Programm immer dann an, wenn ein zur Zeit nicht aktives Fenster zum aktuellen erklärt werden soll. Der Anwender erreicht das bekanntlich durch Klicken in das nicht aktive Fenster. Der AES-Aufruf »WIND_SET« mit der Funktionsnummer 10 und dem Handle, welches das GEM dem Programm übergab, setzt das Fenster an die erste Stelle. Die Variablen »n&« haben keine Bedeutung, übersetzt heißen Sie schlicht »nichts«.

```
> PROCEDURE fenster_schliessen
  LOCAL x&,y&,b&,h&
  fm&=BCLR(fm&,ha%)                         !Entspr. Bit in Fenstermaske löschen
  full_m&=BCLR(full_m&,ha%)                 !Bit der Full-Maske löschen
  ~WIND_GET(ha%,5,x&,y&,b&,h&)              !Ist-Größe holen
  ~GRAF_SHRINKBOX(0,0,0,0,x&,y&,b&,h&)
  ~WIND_CLOSE(ha%)                          !Fenster vom Bildschirm entfernen
  ~WIND_DELETE(ha%)                         !Handle wieder freigeben
  bs&(ha%)=0                                !Löschen von gemerkter Spalte &
  bz&(ha%)=0                                !Zeile
RETURN
```

Hat der Anwender den Closer des Fensters angeklickt, verzweigt das Programm zu dieser Prozedur. Zunächst wird das korrespondierende Bit der Fenster- und der Vollmaske gelöscht. Danach holt sich das Programm über »WIND_GET« die Gesamtgröße des Fensters. Die Funktion »GRAF_SHRINKBOX« zeichnet ein kleiner werdendes Rechteck auf den Bildschirm. Sie hat nur für die Optik des Programms Bedeutung, dient also nicht seiner Funktion. Nach dem Zeichnen wird das Fenster geschlossen, verschwindet. Der Aufruf »WIND_DELETE« entfernt das Fenster aus der AES-Verwaltung, das Handle steht für später zu öffnende Fenster erneut zur Verfügung. Aus diesem Grund müssen zum Schluß auch noch die Koordinaten der linken oberen Ecke des Grafikbildschirms dieses Fensters gelöscht werden.

Das Anklicken des Vollfeldes hat den Aufruf der untenstehenden Procedure zur Folge.

```
PROCEDURE voll_feld
  LOCAL x&,y&,b&,h&,x1&,y1&,b1&,h1&
  IF NOT BTST(full_m&,ha%)                  !Fenster vergrößern
    full_m&=BSET(full_m&,ha%)               !Bit der Full-Maske setzen
    ~WIND_GET(ha%,5,x&,y&,b&,h&)            !Ist-Größe holen
    ~WIND_GET(ha%,7,x1&,y1&,b1&,h1&)        !Größtmögliche Größe holen
    x_a&(ha%)=x&                            !Alte Größe merken
    y_a&(ha%)=y&
    b_a&(ha%)=b&
    h_a&(ha%)=h&
```

```
~WIND_SET(ha%,5,x1&,y1&,b1&,h1&)      !Größtmögliche Größe setzen
~WIND_GET(ha%,4,x&,y&,b&,h&)          !Größe des Arbeitsbereichs
```

Die Variable »full_m&« fungiert ebenfalls als Maske. Jedes gesetzte Bit steht für das entsprechende, auf volle Größe vergrößerte Fenster. Ist das entsprechende Bit nicht gesetzt, heißt das also, daß das Fenster noch nicht über das Vollfeld vergrößert wurde. Es passiert dann folgendes:

Nach dem Setzen besagter Bits erfragt das Programm über entsprechende WIND_GET-Aufrufe die jetzige und danach die größtmögliche Größe. Die Größen hat das Programm zwar schon an anderer Stelle (beim Anlegen des Fensters) erfragt bzw. festgelegt, sie wurden aber nicht gespeichert. Wir hätten sonst zwei weitere Arrays gebraucht, und das Programm wäre unübersichtlicher geworden. Der Geschwindigkeitsnachteil hält sich bei erneuter Nachfrage in Grenzen. Ein weiterer Vorteil besteht darin, daß die Unterprogramme so recht einfach übertragbar sind. Wem das dennoch nicht gefällt, kann das Programm ja ändern. Der WIND_SET-Aufruf vergrößert das Fenster schließlich auf seine maximal mögliche Größe.

```
  ELSE                                  !Fenster zurückführen
    full_m&=BCLR(full_m&,ha%)           !Bit der Full-Maske löschen
    x&=x_a&(ha%)                        !Alte Werte zurück
    y&=y_a&(ha%)
    b&=b_a&(ha%)
    h&=h_a&(ha%)
    ~WIND_SET(ha%,5,x_a&(ha%),y_a&(ha%),b_a&(ha%),h_a&(ha%))
  ENDIF
  ~WIND_GET(ha%,4,x&,y&,b&,h&)          !Größe des Arbeitsbereichs
```

Ist das Fensterbit bei Aufruf des Unterprogramms gesetzt, heißt das, daß das Fenster auf die ursprüngliche Größe zurückzuführen ist. Vorher löscht das Programm das gesetzte Bit. Der Aufruf WIND_SET veranlaßt das AES, das Fenster mit seinen alten Koordinaten und Ausmaßen wieder darzustellen.

```
  bz&(ha%)=MIN(maxh&-h&,bz&(ha%))       !Bildzeile neu berechnen
  bs&(ha%)=MIN(maxb&-b&,bs&(ha%))       !Bildspalte neu berechnen
  GOSUB horizontalen_slider_setzen(x&,y&,b&,h&)
  GOSUB vertikalen_slider_setzen(x&,y&,b&,h&)
RETURN
```

Egal, ob das Fenster vergrößert oder verkleinert wurde, das Programm muß auf jeden Fall die *Größe* und *Position* der Slider neu berechnen und diese dann setzen. Das erledigen die obenstehenden Programmzeilen. Die ersten beiden Zeilen verhindern, daß Bildschirmzeilen bzw. -spalten, die außerhalb des Grafikbildschirms liegen, im Fenster dargestellt werden. Die Unterprogramme, die für das Setzen der Slider verantwortlich zeichnen, stellen wir Ihnen jetzt vor, zunächst aber noch ein paar Erklärungen:

Die Größe eines Sliders ist in den Grenzen von 1 bis 1000 veränderbar. Der Wert 1 stellt dabei die kleinste, der Wert 1000 die maximale Größe eines Sliders dar. Bei der Sliderposition ist es ähnlich. Jeder Slider kann eine Position zwischen 1 und 1000 einnehmen. 1 entspricht dabei der Position ganz oben beim vertikalen Slider oder ganz links beim horizontalen. Hieraus resultiert, daß der Wert 1000 also entweder die Position ganz unten oder ganz rechts repräsentiert.

Slidergröße und -position sind dabei immer relativ zur Fenstergröße zu betrachten. Also: Je größer das Fenster, desto größer sind der Slider und seine Position näher an der Position »ganz unten« bzw. »ganz rechts«. Größe und Position hat das Programm zu setzen. Bei einer Bewegung der Slider teilt uns AES lediglich die gewünschte neue relative Position mit.

Die folgenden beiden Unterprogramme setzen die Größen und Positionen des vertikalen und horizontalen Sliders neu.

```
> PROCEDURE vertikalen_slider_setzen(x&,y&,b&,h&)
  v_o=1000/MAX(maxh&-h&,1)            !Vertikaler Offset
  vslp&=v_o*bz&(ha%)                  !Vertik. Sliderpos. berechnen
  vslg&=1000*h&/maxh&                 !Vertik. Slidergr. berechnen
  ~WIND_SET(ha%,9,vslp&,0,0,0)        !Vertik. Sliderpos. setzen
  ~WIND_SET(ha%,16,vslg&,0,0,0)       !Vertik. Slidergr. setzen
RETURN
```

Wir erklären das Ganze anhand der Positionierung und Größeneinstellung des vertikalen Sliders etwas genauer.

Aus der maximal möglichen Position des Sliders (1000) und der maximal möglichen Y-Koordinate des Grafikbildschirms minus der aktuellen Höhe des Fensters berechnet das Programm einen vertikalen Offset. Er ist in der Variablen »v_o« zu finden. Die Position errechnet sich aus: Offset mal aktuelle Zeile des Grafikbildschirms. Die Größe des Sliders läßt sich aus seinem Maximum (1000), der aktuellen Fensterhöhe und der maximal möglichen Y-Koordinate des Grafikbildschirms berechnen. Die beiden WIND_SET-Aufrufe setzen den Slider danach auf die berechnete Position und Größe. Das Neuzeichnen des Sliders übernimmt das AES.

Alles oben Gesagte gilt sinngemäß auch für die horizontale Sliderposition und seine -größe. Das entsprechende Unterprogramm sieht so aus:

```
> PROCEDURE horizontalen_slider_setzen(x&,y&,b&,h&)
  h_o=1000/MAX(maxb&-b&,1)            !Horizontaler Offset
  hslp&=h_o*bs&(ha%)                  !Horiz. Sliderpos. berechnen
  hslg&=1000*b&/maxb&                 !Horiz. Slidergr. berechnen
  ~WIND_SET(ha%,8,hslp&,0,0,0)        !Horiz. Sliderpos. setzen
  ~WIND_SET(ha%,15,hslg&,0,0,0)       !Horiz. Slidergr. setzen
RETURN
```

Der errechnete horizontale Offset befindet sich in der Variablen »h_o«. Mit zwei anderen WIND_SET-Aufrufen setzt nach der Berechnung das Programm die Position und Größe des horizontalen Sliders neu.

Wir machen mit den anderen Nachrichten weiter, die uns das Betriebssystem zukommen läßt.

Sie können den Fensterinhalt auf vielerlei Weise verschieben. Das Scrollen des Inhalts, mit Ausnahme der Möglichkeit durch Verschiebung der Slider, erledigt die folgende Routine.

Anklicken des Pfeils nach oben schiebt den Inhalt um eine Zeile nach unten, ein Klick auf den Pfeil nach unten schiebt ihn um eine Zeile nach oben. Pfeil nach links bedeutet, daß der Inhalt eine Spalte nach rechts, Pfeil nach rechts, daß er eine Spalte nach links geschoben wird. Klicken in die rechten gepunkteten Randbereiche hat das Verschieben um eine Fensterhöhe nach unten oder nach oben, klicken in die unteren Randbereiche das Verschieben um eine Fensterbreite nach links oder rechts zur Folge.

```
> PROCEDURE scrollen_1
  LOCAL x&,y&,b&,h&
  ~WIND_GET(ha%,4,x&,y&,b&,h&)             !Größe des Arbeitsbereichs
  SELECT MENU(5)                           !Wohin scrollen?
  CASE 0                                   !Seite nach oben
    bz&(ha%)=MIN(MAX(bz&(ha%)-h&,0),maxh&-h&)
```

Für die Berechnungen braucht das Programm die Größe des Arbeitsbereichs. Sie wird durch »WIND_GET« geliefert. Die Richtung, in die gescrollt werden soll, erfährt das Unterprogramm aus dem fünften Eintrag des Message-Buffers. Das Programm hat jetzt dafür zu sorgen, daß kein Bereich kleiner als die Koordinaten (0,0) und keiner größer als die festgelegte Breite und Höhe des Grafikspeichers angezeigt wird. Da die Berechnungen alle ähnlich ablaufen, soll zum besseren Verständnis nur ein Beispiel herausgegriffen werden.

Das Fenster besitzt eine Breite von 300 und eine Höhe von 200 Punkten. In ihm wird der Ausschnitt des Grafikbildschirms mit den Koordinaten 100,100 (entspricht der linken oberen Ecke des Fensters) dargestellt. In den Variablen »bz&(ha%)« und »bs&(ha%)« ist also jeweils der Wert 100 zu finden. Die maximale Höhe und Breite des Grafikbildschirms betrage jeweils 1000 Punkte. Der Anwender möchte den Inhalt des Fensters um eine Fensterhöhe nach unten schieben. Setzen wir einmal die bekannten Werte in die Formel:

```
bz&(ha%)=MIN(MAX(bz&(ha%)-h&,0),maxh&-h&) ein.
```

Wir erhalten:

```
bz&(ha%)=MIN(MAX(100-200,0),1000-200)
```

Auflösung der inneren Klammer ergibt: –100,0; das Maximum davon also 0. Die Formel lautet jetzt:

```
bz&(ha%)=MIN(0,1000-200)
```

Das Minimum von 0 und 800 ist 0, auf unser Beispiel bezogen heißt das, daß der Bildschirm lediglich um 100 Punkte nach oben verschoben werden kann. Fällt Ihnen jetzt das Verstehen der anderen Formeln noch schwer, können Sie nach beschriebener Vorgehensweise fiktive Werte wählen und in die entsprechende Formel einsetzen. Zurück zum Listing:

```
  CASE 1                                 !Seite nach unten
    bz&(ha%)=MIN(bz&(ha%)+h&,maxh&-h&)
  CASE 2                                 !Zeile nach oben
    bz&(ha%)=MIN(MAX(bz&(ha%)-1,0),maxh&-h&)
  CASE 3                                 !Zeile nach unten
    bz&(ha%)=MIN(bz&(ha%)+1,maxh&-h&)
  CASE 4                                 !Seite nach links
    bs&(ha%)=MIN(MAX(bs&(ha%)-b&,0),maxb&-b&)
  CASE 5                                 !Seite nach rechts
    bs&(ha%)=MIN(bs&(ha%)+b&,maxb&-b&)
  CASE 6                                 !Spalte nach links
    bs&(ha%)=MIN(MAX(bs&(ha%)-1,0),maxb&-b&)
  CASE 7                                 !Spalte nach rechts
    bs&(ha%)=MIN(bs&(ha%)+1,maxb&-b&)
  ENDSELECT
  SELECT MENU(5)
  CASE 0,1,2,3
    GOSUB vertikalen_slider_setzen(x&,y&,b&,h&)
  CASE 4,5,6,7
    GOSUB horizontalen_slider_setzen(x&,y&,b&,h&)
  ENDSELECT
  GOSUB inhalt_restaurieren(x&,y&,b&,h&,bs&(ha%),bz&(ha%))
RETURN
```

Nach Berechnung der neu darzustellenden Koordinaten des Grafikbildschirms ruft das Programm bei einer gewünschten vertikalen Verschiebung die Prozedur »vertikalen_slider_setzen«, bei einer horizontalen Verschiebung die Prozedur »horizontalen_slider_setzen« auf. Die Inhaltsrestaurierung muß vom Programm aus erfolgen. Das ist nicht weiter schlimm, es ist die gleiche Routine wie beim Neuzeichnen von Bildschirmteilen und damit bereits vorhanden.

Folgende Prozedur ermöglicht das Scrollen über das Verschieben der Slider. Die Vorgehensweise ist ähnlich wie beim oben besprochenen Scrollen.

```
> PROCEDURE scrollen_2
  LOCAL x&,y&,b&,h&
  ~WIND_GET(ha%,4,x&,y&,b&,h&)              !Größe des Arbeitsbereichs
  SELECT MENU(1)                            !Welcher Slider?
  CASE 25                                   !Horizontaler Slider
```

```
    bs&(ha%)=MENU(5)/h_o
    GOSUB horizontalen_slider_setzen(x&,y&,b&,h&)
  CASE 26                                        !Vertikaler Slider
    bz&(ha%)=MENU(5)/v_o
    GOSUB vertikalen_slider_setzen(x&,y&,b&,h&)
  ENDSELECT
  GOSUB inhalt_restaurieren(x&,y&,b&,h&,bs&(ha%),bz&(ha%))
RETURN
```

Während man bei der ersten Scroll-Routine die Sliderposition selbst errechnete, wird sie nach dem Verschieben der Slider vom AES zurückgeliefert. Das Programm hat also von der Position der Slider auf die Koordinaten des darzustellenden Bildschirmbereichs zu schließen. Die Variablen »h_o« und »v_o« enthalten, wie schon besprochen, einen vertikalen bzw. horizontalen Offset. Am Ende des Unterprogramms wird dann wieder der Inhalt des Fensters auf den aktuellen Stand gebracht.

Jedes Fenster kann vergrößert oder verkleinert werden. Die folgende Prozedur übernimmt diesen Part.

```
> PROCEDURE fenster_groesse
  LOCAL x&,y&,b&,h&
  full_m&=BCLR(full_m&,ha%)                    !Bit der Full-Maske löschen
  ~WIND_SET(ha%,5,MENU(5),MENU(6),MENU(7),MENU(8))
  ~WIND_GET(ha%,4,x&,y&,b&,h&)                 !Größe des Arbeitsbereichs
  bz&(ha%)=MIN(maxh&-h&,bz&(ha%))              !Bildzeile neu berechnen
  bs&(ha%)=MIN(maxb&-b&,bs&(ha%))              !Bildspalte neu berechnen
  GOSUB horizontalen_slider_setzen(x&,y&,b&,h&)
  GOSUB vertikalen_slider_setzen(x&,y&,b&,h&)
RETURN
```

Das AES liefert nach dem Vergrößern oder Verkleinern in den Einträgen 5 bis 8 des Message-Buffers die neuen gewünschten Ausmaße zurück. Das Programm setzt zunächst über »WIND_SET« das Fenster auf die gewünschte Größe und erfragt dann die Größe des Arbeitsbereichs mit »WIND_GET«. Danach berechnet es die linke obere Ecke des darzustellenden Ausschnitts vom Grafikbildschirm neu. Die dann folgenden Unterprogrammaufrufe passen Größe und Position der Slider den neuen Gegebenheiten an.

Wir kommen zur letzten Nachricht, die das AES uns bei Manipulationen der Fenster zukommen lassen kann.

```
> PROCEDURE fenster_bewegung
  full_m&=BCLR(full_m&,ha%)                    !Bit der Full-maske löschen
  ~WIND_SET(ha%,5,MENU(5),MENU(6),MENU(7),MENU(8))
RETURN
```

Jedes Fenster kann auf dem Desktop eine beliebige Position einnehmen. Es ist sogar möglich, es nach rechts oder nach unten aus dem Bildschirm hinauszuschieben. Die neuen Koordinaten stehen danach in den Einträgen 5 bis 8 des Message-Buffers. Das Programm hat nichts anderes zu tun, als über einen WIND_SET-Aufruf das Fenster auf eben diese Position zu setzen. Wird das Neuzeichnen einiger Bildschirmteile erforderlich, meldet sich das AES auf altbekannte Art und Weise.

Wir kommen zu den letzten Unterprogrammen. Sie sind für das Zeichnen, das Retten des Fensterinhalts und das Beenden des Programms zuständig.

```
> PROCEDURE zeichnen
  LOCAL x&,y&,b&,h&,x1&,y1&,x2&,y2&,ha%
  ~WIND_GET(ha%,10,ha%,n&,n&,n&)            !Aktuelles Fenster abfragen
~WIND_GET(ha%,4,x&,y&,b&,h&)                !Größe Arbeitsbereich
  DEC b&
  DEC h&
```

Drückt der Benutzer die linke Maustaste außerhalb der Bedienelemente, verzweigt das Programm zum Zeichnen. Zuerst besorgt sich die Routine die Nummer des aktuellen Fensters und die Größe seines Arbeitsbereichs. Das Dekrementieren der Breite und der Höhe ist erforderlich, da wir als Koordinaten die Umrandungslinie erhalten.

```
  IF MOUSEX<x&+b& AND MOUSEY<y&+h&
    CLIP x&,y& TO x&+b&,y&+h&
    GRAPHMODE 3
    x1&=MAX(MOUSEX,x&)
    y1&=MAX(MOUSEY,y&)
```

Befindet sich der Mauscursor innerhalb des Fensters, wird zunächst das Clipping-Rechteck auf Arbeitsbereichsgröße gesetzt und dann der XOR-Grafikmodus eingeschaltet. Die nächsten beiden Zeilen legen die Ursprungskoordinaten des Rechtecks fest.

```
    DO
      x2&=MIN(MAX(MOUSEX,x1&),x&+b&)
      y2&=MIN(MAX(MOUSEY,y1&),y&+h&)
      BOX x1&,y1&,x2&,y2&
      BOX x1&,y1&,x2&,y2&
      EXIT IF MOUSEK=2
    LOOP
```

Die Schleife zeichnet eine etwas flimmernde Box in das Fenster. Ihre Größe hängt von der jeweiligen Mausposition ab. Drücken der rechten Maustaste bewirkt das Verlassen der Schleife (die Box ist nicht mehr sichtbar).

```
    IF x1&<x2& AND y1&<y2&
      GRAPHMODE 1
      BOX x1&,y1&,x2&,y2&
      GOSUB fensterinhalt_retten
    ENDIF
  ENDIF
RETURN
```

Sind Anfangs- und Endkoordinaten verschieden, schaltet das Programm in den Überschreibmodus zurück und zeichnet die erzeugte Box in das Fenster. Die Änderung des Fensterinhalts wird mit dem Prozeduraufruf »fensterinhalt_retten« in den Grafikbildschirm übernommen. Danach ist alles erledigt, das Programm kehrt zur Warteschleife zurück.

Kommen wir zum Retten des veränderten Fensterinhalts. Die Hauptfunktion übernimmt wieder die bekannte BITBLT-Routine.

```
> PROCEDURE fensterinhalt_retten
  LOCAL x&,y&,b&,h&,mx&,my&,mb&,mh&
  ~WIND_GET(0,5,mx&,my&,mb&,mh&)        !Größe des Desktops ermitteln
  ~WIND_GET(ha%,4,x&,y&,b&,h&)          !Größe des Arbeitsbereichs
  IF x&+b&>mb&                          !Arbeitsbereich voll sichtbar?
    b&=mb&-x&
  ENDIF
  IF y&+h&>mh&
    h&=mh&-y&
  ENDIF
```

Die ersten Zeilen des Unterprogramms verhindern, daß Teile des Fensters, die außerhalb des Desktops liegen, mitkopiert werden. Fehlen die Zeilen, dann ist ein falscher Inhalt des Grafikspeichers nach dem Kopiervorgang sicher.

```
  HIDEM                                 !Maus aus
```

Der Mauscursor muß unbedingt ausgeschaltet werden, da er sonst ebenfalls in den Grafikspeicher kopiert wird.

Den folgenden Teil brauchen wir wohl nicht mehr zu erklären, er ist mit dem entsprechenden Teil (bis auf die vertauschten Adressen) der Neuzeichen-Routine identisch.

```
  bb%(0)=b&                             !Breite des Rasters der Quelle
  bb%(1)=h&                             !Höhe des Rasters der Quelle
  bb%(5)=&H3030303                      !Verknüpfungsmodus (Ziel=Quelle)
  bb%(6)=x&                             !X-Koordinate Quelle
  bb%(7)=y&                             !Y-Koordinate Quelle
  bb%(8)=akt_bs%                        !Adresse der Quelle
  bb%(10)=80                            !Offset zur nächsten Zeile (von) in Byte
  bb%(12)=bs&(ha%)                      !X-Koordinate Ziel
```

```
  bb%(13)=bz&(ha%)                    !Y-Koordinate Ziel
  bb%(14)=ad_bs%                      !Adresse des Ziels
  bb%(16)=(((maxb&/16) AND 65534)+2)*2 !Offset zur nächsten Zeile (Ziel)
  BITBLT bb%()                        !Bit-Block verschieben
  SHOWM                               !Maus wieder ein
RETURN
```

Das nächste Unterprogramm ist, so unglaublich es klingt, das wichtigste unseres Programms. Es wurde von uns zuerst programmiert und stellt das ordnungsgemäße Beenden des Programms auch bei Fehlern sicher.

```
> PROCEDURE ende
  FOR i&=1 TO 7                       !Alle Fenster schließen, Handle freigeben
    IF BTST(fm&,i&)
      ~WIND_CLOSE(i&)
      ~WIND_DELETE(i&)
    ENDIF
  NEXT i&
```

Zunächst schließt und entfernt das Programm in der Schleife jedes Fenster, dessen korrespondierende Bit in der Fenstermaske gesetzt ist.

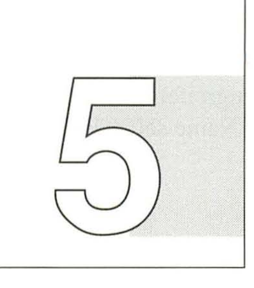
```
  MENU KILL                           !Menüabfrage aus
  CLIP 0,0,640,400                    !Clip auf vollen Bildschirmbereich
  CLS
```

Das Pull-down-Menü wird entfernt (kann nicht mehr aufgerufen werden), das Clipping-Rechteck auf die volle Bildschirmgröße gesetzt und der Bildschirm gelöscht.

```
  ~MFREE(speicher_ad%)                !Speicher an Betriebssystem zurück
  RESERVE                             !Gesicherten Speicher an GFA zurück
  END
RETURN
```

Zum Schluß gibt das Programm den reservierten Speicher an das GEM-DOS zurück. Er wird danach durch »RESERVE« erneut dem GFA-Interpreter zugeteilt, der Anfangszustand ist wieder erreicht.

Wir hoffen, daß Sie nach allen Erklärungen nicht die Lust an der Fensterprogrammierung verloren haben. Wer dieses Kapitel aufmerksam verfolgte, hat vielleicht festgestellt, daß diese Art der Programmgestaltung zwar recht aufwendig, aber gar nicht so schwierig ist. Auch wir mußten uns mit dem zu treibenden Aufwand erst anfreunden. Seitdem aber die Grundroutinen (Sie haben sie gerade kennengelernt) programmiert sind, fällt es uns sehr leicht, die Bildschirmausgaben in einem Fenster vornehmen zu lassen. Also, nur Mut, schreiben Sie Ihre zukünftigen Programme bedienerfreundlich unter GEM!



Diskettenbefehle

Diskstation oder Festplatte sind nach dem Monitor die wichtigsten Peripheriegeräte des ST. Daher beziehen sich auch sehr viele Befehle des GFA-Basic auf sie. Dieser Teil des Buchs beschreibt die wichtigsten von ihnen; eine komplette Übersicht finden Sie im Anhang des Buchs. Wir möchten Sie an dieser Stelle um etwas Geduld bitten, da das nächste Programm, eine Artikelverwaltung für Zeitungen, erst am Ende des Kapitels folgt. Weil es nach unserer Meinung sehr wichtig ist, den Umgang mit dem verwendeten Speichermedium oder auch angeschlossenen Peripheriegeräten sicher zu beherrschen, folgt zunächst eine längere Einweisung in diese Materie.

Der ST kann mehrere Laufwerke verwalten. Jedem Laufwerk wird bei seiner Anmeldung über das Desktop vom Anwender ein *Kennbuchstabe* von A bis P zugewiesen. Die Buchstaben A und B sind Floppy-Disk-Stationen vorbehalten. Die anderen Buchstaben werden an Partitionen einer Festplatte oder an eine RAM-Disk vergeben. Jedes Laufwerk bzw. jede Partition ist daher durch den zugehörigen Kennbuchstaben klar definiert. Damit eine bestimmte Datei von einem bestimmten Laufwerk geladen werden kann, bedarf es noch mindestens einer weiteren Information, dem Dateinamen. Er besteht aus einem bis zu acht Zeichen langen Namen und einer bis zu drei Zeichen langen Extension. Getrennt wird beides durch einen Dezimalpunkt.

Zur übersichtlicheren Gestaltung des Inhaltsverzeichnisses können auch *Ordner* angelegt werden. Für die Benennung der Ordner gilt das gleiche wie für die Dateinamen. Jeder Ordner hat wieder sein eigenes Inhaltsverzeichnis. Hinter diesem verbirgt sich oft eine ganze Gruppe von zusammengehörigen Dateien und/oder einem oder mehreren weiteren Ordnern, die wieder Unterordner haben dürfen. Wie gesagt, das kann, muß aber nicht so sein. Wie der Anwender sein Inhaltsverzeichnis gestaltet, bleibt ihm selbst überlassen. Die Verschachtelung von Ordnern nennt man auch *»hierarchisches File-System«*.

Für den Zugriff auf eine Datei, die in einem bestimmten Ordner eines bestimmten Laufwerks zu finden ist, benötigt man den sogenannten *Suchpfad*. Dabei ist noch zwischen dem *aktuellen* Suchpfad und einem *definierten* Suchpfad zu unterscheiden. Der aktuelle Suchpfad ist in der Regel der, aus dem das Programm gestartet wurde. Damit das alles leichter zu verstehen ist, folgt ein Beispiel:

An den ST sollen zwei Laufwerke mit den Kennbuchstaben »A:« und »B:« angeschlossen sein. Das aktuelle Laufwerk sei »A:«. Wir wollen auf eine Datei zugreifen, die sich auf der Diskette im Laufwerk »B:« im Ordner »TEST.ORD« befindet, ihr Name sei »TEST.DAT«. Als Suchpfad müssen wir dem Rechner den folgenden übergeben:

```
B:\TEST.ORD\TEST.DAT
```

Die Buchstaben dürfen auch klein geschrieben werden, den ST stört das nicht.

Zur Erklärung:

B:	Diese beiden Zeichen bezeichnen das Laufwerk.
\TEST.ORD	Durch den umgekehrten Schrägstrich (auch Backslash genannt) wird der Ordnername von der Laufwerksbezeichnung getrennt.
\TEST.DAT	Dieser Schrägstrich trennt den Ordnernamen vom Dateinamen.

Für den Dateinamen stehen noch zwei besondere Zeichen zur Verfügung: »?« und »*«. Das Fragezeichen darf für jedes andere Zeichen im Dateinamen stehen. Auf unser Beispiel bezogen, könnte man auch »????.DAT« oder »T?ST.D??« usw. schreiben. Zu beachten ist dabei, daß evtl. das verkehrte File angesprochen wird. Existiert zum Beispiel auch die Datei »1111.DAT«, so wählt sie der Rechner bei Angabe von »????.DAT« statt der Datei »TEST.DAT« aus.

Das Multiplikationszeichen teilt dem Rechner mit, daß dem Anwender alle folgenden Zeichen »egal« sind. Ist lediglich eine Datei vorhanden, reicht folglich »*.*«. Soweit zu den Suchpfaden, weitere Beispiele zu dem Thema finden Sie bei den Erklärungen der Befehle.

Einige der Diskettenbefehle erlauben auch den Zugriff auf die Schnittstellen des ST und die dort angeschlossenen Geräte. Man übergibt dann anstatt des Suchpfads eine Abkürzung des Schnittstellennamens. Untenstehende Bezeichnungen sind zugelassen:

Name	**Schnittstelle**
LST oder PRN	Parallele Schnittstelle (Centronics), an ihr wird in den meisten Fällen der Drucker angeschlossen sein.
CON	Console, d.h. Monitor. Alle Steuerzeichen wie ESC werden ausgeführt.
VID	Console bzw. Monitor. Die Steuerzeichen werden angezeigt.
AUX	Serielle Schnittstelle.
MID	MIDI-Schnittstelle.
IKB	Schreibzugriff auf den Tastaturprozessor. Dabei ist größte Vorsicht angeraten. Bei falschen Daten verabschiedet sich der Rechner.

Wenden wir uns jetzt den wichtigsten Diskettenbefehlen zu.

Mit den ersten beiden Befehlen gibt GFA ein Inhaltsverzeichnis aus. Zu beachten ist, daß jeder Befehl nur den Inhalt des Hauptverzeichnisses oder eines Ordners ausgibt.

DIR Pfad$ oder
DIR Pfad$ TO Dateiname$

Bei Verwendung der ersten Syntax erhalten Sie die Dateinamen mit der Extension auf dem Bildschirm zurück. Mit Hilfe der zweiten ist es möglich, die Ausgabe in eine Datei oder auf ein Peripheriegerät zu lenken.

Pfad$	In dieser Variablen ist der Suchpfad zum gewünschten Inhaltsverzeichnis definiert.
Dateiname$	Enthält Suchpfad und Dateinamen oder den Namen des Peripheriegeräts.

Beispiele:

DIR "A:\TEST.*" stellt alle Dateinamen des Laufwerks »A:«, die im Wurzelverzeichnis stehen und den Namen »TEST« und eine beliebige Extension besitzen, auf dem Bildschirm dar.

DIR "A:*.*" TO "A:\INHALT.DAT" schreibt den Inhalt des Wurzelverzeichnisses des Laufwerks »A:« in eine Datei namens »INHALT.DAT«.

DIR "A:*.*" TO "LST:" gibt den gleichen Inhalt auf dem Drucker aus.

Der nächste Befehl gibt die Dateinamen neben der Extension, Uhrzeit, Länge und dem Datum eines in »Pfad$« festgelegten Verzeichnisses aus.

FILES Pfad$ oder
FILES Pfad$ TO Dateiname$

Beispiele:

FILES "A:*.PRG" zeigt alle Dateien des Wurzelverzeichnisses von Laufwerk »A:« mit der Extension ».PRG« auf dem Bildschirm an.

FILES "\T???.DOC". Alle Dateien mit der Extension ».DOC«, und deren Name vier Zeichen lang ist und mit dem Buchstaben »T« beginnt, werden ausgegeben. Der Zugriffspfad bezieht sich dabei auf das Wurzelverzeichnis des aktuellen Laufwerks.

Wie man sich einen Überblick über die auf der Diskette vorhandenen Dateien verschafft, wissen Sie jetzt. Leider lassen sich die Inhaltsverzeichnisse nicht ohne Umwege von einem Programm in Variablen lesen. In den meisten Fällen reicht es aber, zu wissen, ob eine Datei bereits vorhanden ist. Dafür gibt es folgende Funktion:

A&=EXIST(Pfad$)

»Pfad$« enthält wieder den gewünschten Zugriffspfad. Ist die Datei vorhanden, erhält man in der Variablen »A&« den Wert –1 (True) zurück. Existiert die Datei nicht, enthält »A&« den Wert 0 (False).

Beispiel:

PRINT EXIST("A:*.*") ergibt True, wenn die Diskette im Laufwerk »A:« mindestens eine Datei enthält, oder False, wenn sie leer ist. Das funktioniert aber nur, wenn die Diskette formatiert ist.

Dateien können auch umbenannt werden. GFA macht selbst auch davon Gebrauch, wenn eine bereits vorhandene Programmdatei erneut gespeichert wird. Die alte Datei erhält dann die Extension ».BAK«. Der Befehl heißt:

NAME Alter_name$ AS Neuer_name$ oder
RENAME Alter_name$ AS Neuer_name$

In der Variablen »Alter_name$« stehen Suchpfad und Name der Datei, deren Name geändert werden soll. Die Variable »Neuer_name$« enthält den neuen Suchpfad und den neuen Namen, den die Datei erhalten soll. Eine Änderung des Zugriffpfads ist also auch möglich. Der Befehl kann nur innerhalb desselben Laufwerks angewandt werden.

Das Löschen einer Datei geschieht mit dem Befehl:

KILL Pfad_name$

In der Variablen »Pfad_name$« stehen Suchpfad und Name der zu löschenden Datei. Seien Sie bei der Verwendung vorsichtig, die Restaurierung eines einmal gelöschten Files kann viel Zeit kosten oder ist manchmal sogar ganz unmöglich.

Bei einer Dateiverwaltung ist es meistens recht sinnvoll, die erzeugten Dateien alle in einem Ordner abzuspeichern. Damit der Anwender den Ordner nicht umständlich über das Desktop selbst erzeugen muß, überläßt man diese Arbeit dem Programm. Dazu dient der folgende Befehl:

MKDIR Ordnername$

Die Stringvariable »Ordnername$« enthält den Suchpfad.

Beispiel:
MKDIR "\GFA.DAT" legt auf dem aktuellen Laufwerk im Wurzelverzeichnis einen Ordner namens »GFA.DAT« an.

Enthält ein Ordner keine Dateien, läßt er sich auch wieder löschen. Der Befehl:

RMDIR Ordnername$

Beispiel:
RMDIR "\GFA.DAT" löscht einen vorhandenen, leeren Ordner namens GFA.DAT.

Die folgenden Befehle benötigen statt eines Kennbuchstabens für die Laufwerke eine Kennziffer. Die Kennziffer 0 bedeutet, daß sich der Befehl auf das aktuelle Laufwerk bezieht. Die Zahlen 1 bis 16 bezeichnen die möglichen Laufwerke. Die Zuordnung ist wie folgt: Das Laufwerk »A:« hat die Kennziffer 1, Laufwerk »B:« die Kennziffer »2«, ..., Laufwerk »P:« schließlich die Kennziffer 16.

Vor dem Speichern von Dateien sollte immer überprüft werden, ob der Platz auf der Diskette reicht. Mit der folgenden Funktion ist das leicht zu bewerkstelligen:

L%=DFREE(Lw&)

Die Variable »Lw&« enthält die Kennziffer des Laufwerks, die Variable »L%« nach dem Aufruf den noch verfügbaren Platz des Speichermediums.

Viele Anwendungen benötigen den aktuellen Suchpfad. Auch dafür gibt es eine Funktion:

Pfad$=DIR$(Lw&)

In der Stringvariablen »Pfad$« steht nach dem Aufruf der eingestellte Suchpfad des Laufwerks »Lw&«. Ist die Stringvariable nach dem Aufruf leer, bezieht sich der eingestellte Suchpfad auf das Wurzelverzeichnis.

Beispiel:
GFA-Basic wurde vom Laufwerk »A:« aus dem Ordner »GFA.ORD« aufgerufen. In dem Ordner befindet sich auch unser Programm, das geladen und gestartet wurde. Der Befehl »?DIR$(0)« liefert jetzt »\GFA.ORD\« zurück. Selbstverständlich kann mit dieser Funktion auch der aktuelle Pfad jedes anderen angeschlossenen Laufwerks erfragt werden.

Der aktuelle Suchpfad läßt sich jederzeit ändern. Dazu bedarf es nur des Einsatzes des folgenden Befehls:

CHDIR Pfad$

Der Befehl bezieht sich immer auf das aktuelle Laufwerk.

Beispiel:
Das aktuelle Laufwerk besitzt im Wurzelverzeichnis den Ordner GFA.DAT und dieser den Ordner DATEI.DAT. Durch »CHDIR "\GFA.DAT\DATEI.DAT"« wird der aktuelle Suchpfad in \GFA.DAT\DATEI.DAT geändert. Übergibt man nur »\«, wird das Wurzelverzeichnis eingestellt.

Auch das aktuelle Laufwerk kann mit einem Befehl geändert werden. Dazu darf entweder der Kennbuchstabe oder die Laufwerksnummer übergeben werden. Der Befehl heißt:

CHDRIVE Lw& oder
CHDRIVE L$

Die Aufrufe »CHDRIVE 1« und »CHDRIVE "A:\"« erklären beide das Laufwerk »A:« zum aktuellen Laufwerk. Soviel zu den allgemeinen Diskettenbefehlen.

5.1 Dateiverwaltung

Dieser Abschnitt behandelt ein Thema, dem sich ein ernsthafter Programmierer nicht lange verschließen kann. Früher oder später schreibt man Programme, die Daten sichern oder einlesen müssen. Im einfachsten Fall handelt es sich um Bildschirminhalte, komplizierter wird das Ganze schon beim Aufbau einer Adreßdatei. Bevor wir auf die Dateibefehle näher eingehen, folgt zunächst wieder die Beschreibung wichtiger Zusammenhänge.

5.2 Grundlagen

»Viele Wege führen nach Rom«, dieser Ausspruch läßt sich auch auf die Dateiverwaltung übertragen. Wir stellen Ihnen in den folgenden Abschnitten verschiedene Lösungen der Dateiverwaltung vor, d.h. »gute« und »schlechte«.

Man unterscheidet zwischen zwei Dateienarten. Die erste heißt sequentielle Datei, bei der zweiten handelt es sich um die relative Datei. Sequentiell heißt frei übersetzt »hintereinander«. Diese Dateienart wird in den meisten Fällen innerhalb eines Vorgangs in den Rechner gelesen oder auch gespeichert. Beispiele beim Atari sind Programmdateien. Wir wollen jetzt eine eigene Datei einrichten, dazu gleich ein Beispiel:

Sie speichern die Vornamen Janet, Jürgen und Klaus in der Datei TEST.DAT ab. Nach dem Schließen der Datei (wird später noch erklärt) hat diese, je nach Speicherart, mindestens eine Länge von 16 Byte. Ihr Inhalt sieht, für unser Beispiel, folgendermaßen aus:

```
JanetJürgenKlaus
```

Die Daten stehen also in der gleichen Reihenfolge, in der sie gespeichert wurden. Beim Wiedereinlesen der Daten steht man vor einem nicht zu unterschätzenden Problem: Wie trennt man die Namen wieder, sprich, wie bringt man zum Beispiel die Janet und den Jürgen auseinander? Der Programmierer hat wieder die Wahl zwischen zwei Möglichkeiten.

Die erste Möglichkeit:

Jedem Namen wird vor dem Speichern ein oder mehrere ASCII-Codes angehängt, die bestimmt nie in einem der Namen vorkommen werden. Spielen wir diesen Fall gedanklich durch, das Zeichen sei das mit dem ASCII-Wert 0 (GFA-Basic benutzt bei Verwendung

entsprechender Befehle ASCII 13 und ASCII 10). Dieses Zeichen wird also jedem Namen angehängt. Auf unserer Diskette sieht die Datei nach dem Speichern so aus:

```
Janet0Jürgen0Klaus0
```

Hinweis: Die anderen Buchstaben liegen selbstverständlich ebenfalls als ASCII-Code vor. Wir halten aber diese Art der Darstellung in vorliegendem Fall für übersichtlicher.

Wurde eine Datei nach diesem Schema abgespeichert, kann man sie später byteweise (Zeichen für Zeichen) einlesen und die Namen anhand des Zeichens 0 wieder trennen. Diese Art des Einlesens kostet, gerade bei längeren Dateien, sehr viel Zeit.

Die zweite Möglichkeit:

Vor dem Speichern eines Namens kennen wir seine jeweilige Länge, wenn nicht, läßt sie sich schnell feststellen. Legen wir doch einfach eine zweite Datei namens »TEST_L.DAT« an, die die Länge des jeweiligen Namens enthält. Beim Einlesen der Daten sind wir damit aus dem Schneider. Noch einmal das Beispiel:

Die Datei TEST.DAT enthält wieder die eigentlichen Daten:

```
JanetJürgenKlaus
```

Die zweite Datei TEST_L.DAT enthält die Länge des jeweiligen Namens, in ihr sind folgende drei Einträge zu finden, die Zahlen sind ASCII-Codes:

```
565
```

Wohlgemerkt: In dieser Datei stehen ebenfalls Zeichen. Es handelt sich bei der 5 also nicht um die Zahl Fünf, sondern um das Zeichen mit dem ASCII-Code Fünf. Vor dem Speichern wird der Längenwert mit »CHR$« entsprechend umgewandelt.

Wir lesen zuerst die Datei, die die Länge der Namen enthält, ein. Danach werden immer nur so viele Zeichen aus der Namendatei gelesen, wie uns der entsprechende Wert, umgewandelt durch »ASC«, der Längendatei vorgibt. Darf ein Name im Programm länger als 255 Zeichen sein, so wandelt man die Länge vor dem Speichern mit »MKI$« um. Zurückgewandelt wird sie nach dem Einlesen mit »CVI«. Jeder Name darf dann bis zu 65535 Zeichen lang sein. Mit den anderen Umwandlungsfunktionen geht's sogar noch länger. Achten Sie bitte darauf, daß Sie bei einem einmal gewählten Zahlenformat bleiben. »CHR$« belegt nur ein Byte der Längendatei, »MKI$« aber zwei, andere Formate noch mehr. Bei der Vermischung der Formate ist nichts gewonnen. Einen nicht zu unterschätzenden Nachteil haben wir bei dieser Art der Dateikonstruktion noch nicht beseitigt:

Das Löschen von Datensätzen läßt sich nur mit einigem Aufwand erreichen. Man kann zwar den entsprechenden Namen überschreiben, erhält dann aber, wenn keine entsprechende Datei geführt wird, einen Leerstring beim Einlesen zurück. Eine andere Möglichkeit bestünde darin, die Datei komplett einzulesen, den Namen im Speicher zu löschen und danach die so berichtigte Datei nebst ebenfalls berichtigter Längendatei wieder abzuspeichern. Geht man so vor, darf man sich an zwei Dingen nicht stören: den recht

langen Zugriffszeiten und dem erforderlichen Speicherbedarf des Rechners. Gerade letzteres muß bei sehr großen Dateien berücksichtigt werden, soll doch die Funktion des Programms nicht vom Besitz eines Mega ST4 abhängen. Ein Megabyte auf der Diskette ist wesentlich günstiger zu erstehen als ein Megabyte Speicher. Bei dieser Art der Dateiverwaltung brauchen wir sogar beides, erst einmal den Speicherplatz auf der Diskette und das gleiche noch mal im Rechner. Die Lösung des Problems bietet sich durch Verwendung von relativen Dateien an.

Relativ hat in diesem Fall nichts mit dem berühmten Haar in der Suppe zu tun (ein Haar auf dem Kopf ist relativ wenig, ein Haar in der Suppe relativ viel), sondern bezieht sich hier auf den Dateianfang. Vorraussetzung für eine relative Datei ist immer die gleiche Länge aller Datensätze.

Bleiben wir bei unserem Beispiel. Beim Programmieren unserer Datenerfassung wird festgelegt, daß kein Name länger als zehn Zeichen sein darf. Da jeder Datensatz die gleiche Länge haben muß, wird jeder Name, der die festgelegte Länge unterschreitet, mit Leerzeichen aufgefüllt. Außerdem muß im Programm dafür gesorgt sein, daß ein Datensatz nicht länger als zehn Zeichen ist. Danach sorgen wir noch dafür, daß der Name linksbündig im Datensatz erscheint. Auf unser Beispiel bezogen, sehen die drei Datensätze auf der Diskette jetzt so aus:

```
JanetXXXXXJürgenXXXXKlausXXXXX
```

Der Buchstabe »X« steht für ein Leerzeichen, das Dateiende wäre sonst nicht zu erkennen gewesen.

Auf den ersten Blick hat sich kaum etwas verändert. Die Datei ist etwas länger geworden, und die Einträge stehen immer noch sequentiell auf der Diskette. Durch die Festlegung der Eintragslänge haben wir immerhin die Längendatei oder die Trennzeichen gespart. Außerdem wurden weitere Vorteile erworben. Bevor der Beweis angetreten wird, noch etwas Grundsätzliches. Wir sprachen zwar schon oben von einem Datensatz, das war an der Stelle nicht ganz richtig, da verschiedene Längen vorkommen durften. Hier ist die Bezeichnung dagegen korrekt. Jeder Datensatz hat eine fest definierte Länge. Durch die Längendefinition ist es jetzt dem Rechner intern möglich, jedem Datensatz eine Datensatznummer, in entsprechender Reihenfolge, zuzuteilen.

Der Datensatz »JanetXXXXX« hat die Nummer 1, »JürgenXXXX« die Satznummer 2 und »KlausXXXXX« die Nummer 3. Durch einen speziellen Befehl läßt sich zum Beispiel nur der Datensatz mit der Nummer 3 schreiben, mit einem anderen läßt er sich wieder einlesen. Es folgt eine weitere Stärke dieser Dateienart.

Bei der Dateiverwaltung können die Datensätze meistens in beliebiger Reihenfolge eingegeben werden. Später werden sie oft, geordnet nach einem bestimmten Sortierkriterium, wieder ausgegeben. Im einfachsten Fall wäre es die alphabetische Reihenfolge. Die Lösung des Problems:

Zuerst liest das Programm Datensatz für Datensatz ein (es können auch nur dessen Teile sein, nach denen sortiert werden soll) und sortiert sie entsprechend. Parallel dazu wird im Speicher eine sequentielle Datei aufgebaut. Sie enthält, nach der Sortierung, die Datensatznummern und wird dann ebenfalls abgespeichert. Soll nun auf die Datensätze zugegriffen werden, so lädt man die sequentielle Datei ein und greift über die Datensatznummer auf den entsprechenden Datensatz der relativen Datei zu. Bei einer Adreßdatei wäre es auch vorstellbar, daß man noch nach Postleitzahlen sortiert. Ebenfalls kein Problem, man erhält dann lediglich eine zweite Datei zurück, über die dann wieder auf die Datensätze zugegriffen werden kann. Vorteil bei den genannten Aktionen ist zum einen der wesentlich kürzere Zugriff auf die entsprechenden Datensätze (die jeweilige sequentielle Datei ist wesentlich kürzer als die relative Datei) und zum anderen die Speicherplatzersparnis auf der Diskette.

Vielleicht ist das Ganze noch ein wenig schwer zu verstehen, das ist nicht weiter schlimm. Uns kam es darauf an, Ihnen den Unterschied zwischen den Dateiarten begreiflich zu machen. Das Programm am Ende des Kapitels benutzt beide, die entsprechenden Passagen werden dort ausführlich erklärt.

5.3 Allgemeine Dateibefehle

Vor dem Einrichten oder Verändern einer Datei muß ein Datenkanal geöffnet werden. Für sequentielle Dateien gilt:

OPEN "Modus",Nummer,"Name"

Für relative Dateien:

OPEN "Modus",#Nummer,"Name",Datensatzlänge

Der Parameter Modus:

Modus = "I"	Sequentielle Datei zum Lesen öffnen.
Modus = "O"	Sequentielle Datei zum Schreiben öffnen.
Modus = "U"	Bestehende Datei zum Lesen und Schreiben öffnen.
Modus = "A"	Daten an eine bestehende sequentielle Datei anfügen.
Modus = "R"	Relative Datei zum Lesen und Schreiben öffnen.

Der Modus "I" kann nur gewählt werden, wenn die Datei mit dem entsprechenden Dateinamen bereits existiert. Was sollte man sonst auch lesen? Übergibt man als Modus "O", wird eine schon bestehende Datei zum Überschreiben geöffnet, ist noch keine vorhanden wird sie neu eingerichtet. Der Modus "U" findet Verwendung, wenn eine bestehende Datei für Lese- und Schreibzugriffe geöffnet werden soll. Soll eine sequentielle Datei erweitert werden, verwendet man zweckmäßigerweise den Modus "A".

Nummer — Ist eine Zahl zwischen 0 und 99. Es können also bis zu 100 Dateien gleichzeitig geöffnet werden. Sind mehrere Dateien gleichzeitig offen, darf jede Nummer nur einmal vergeben sein. Über die Kanalnummer wird später die geöffnete Datei angesprochen.

Name — Steht für Dateiname und Suchpfad, wir haben die Zusammenhänge schon hinreichend erklärt. Als Name darf auch hier wieder ein Schnittstellenname verwendet werden, die entsprechende Übersicht haben Sie bereits erhalten.

Soll eine relative Datei geöffnet werden, ist als Modus "R" und außerdem die Datensatzlänge zu übergeben.

Eine einmal geöffnete Datei muß auch wieder geschlossen werden. Mißachtung dieser Regel straft der Rechner mit »Dateileichen« bzw. unvollständigen Einträgen. Das Schließen übernimmt der Befehl:

CLOSE #Nummer oder
CLOSE

Nummer — Enthält die Zahl, die der Datei beim Öffnen zugeordnet wurde. Die zweite Syntax schließt alle geöffneten Dateien.

Wir kommen jetzt zu den Befehlen, die das Lesen und Schreiben ermöglichen. Zum besseren Verständnis noch einige Anmerkungen. Beim Öffnen einer Datei wird ein Dateipointer vom Rechner eingerichtet. Er zeigt zunächst auf das erste Byte bzw. den ersten Datensatz der Datei. Beachten Sie bitte: Der Zeiger hat dann den Wert 0. Ausnahme ist der Modus "A", hier zeigt er auf das letzte Byte. Die Position des Pointers läßt sich mit dem folgenden Befehl erfragen:

P%=LOC(#Nr)

P% — Enthält nach dem Aufruf die Position der vorher geöffneten Datei mit der Kennummer »Nr«. Wir können auch erfragen, ob sich der Pointer am Ende der Datei befindet. Die Funktion:

E%=EOF(#Nr)

Ist das Dateiende erreicht, gibt E% den Wert –1 zurück (TRUE), wenn nicht, erhält man 0 (FALSE). Die Länge einer vorher geöffneten Datei gibt untenstehende Funktion zurück.

L%=LOF(#Nr)

In »L%« steht nach dem Aufruf die Länge der Datei.

Wenn Sie wollen, können Sie Datum und Zeitangabe einer vorher geöffneten Datei jederzeit auf die Systemdaten neu einstellen. Dazu dient der Befehl:

TOUCH #Nr

Nach Aufruf werden die Daten auf den aktuellen Stand gebracht. Bei konsequenter Anwendung erkennt man so immer im Inhaltsverzeichnis, wann zum letzten Mal mit der Datei gearbeitet wurde.

Die nächsten beiden Abschnitte behandeln die speziellen Befehle der beiden Dateienarten.

5.4 Befehle und Funktionen für sequentielle Dateien

Die jetzt folgenden Befehle beziehen sich auf sequentielle Dateien. Sehr nützlich ist es, daß man die Position des Dateipointers beeinflussen kann. Das geht mit untenstehenden Befehlen:

Hinweis: Der Datenzeiger darf nicht über den Dateianfang oder das Dateiende bewegt werden, GFA stoppt sonst den Programmlauf und gibt eine entsprechende Fehlermeldung aus.

SEEK #Nr,Position%

Dieser Befehl setzt den Pointer auf das in der Variablen »Position« angegebene Byte der Datei. Auch die Änderung des Pointers relativ zu seiner derzeitigen Position ist möglich.

RELSEEK #Nr,Anz%

Zu diesem Befehl gleich ein Beispiel:

Der Pointer befinde sich auf dem zwanzigsten Byte der Datei. In der Variablen »Anz%« übergeben wir 5. Der Zeiger ist danach auf Byte 25 gerichtet. Übergibt man eine negative Zahl, zum Beispiel –10, zeigt der Pointer danach auf Byte 10 der Datei. Die Anzahl 0 setzt den Zeiger auf den Dateianfang.

Es folgen Befehle, die Daten in eine vorher geöffnete Datei schreiben.

OUT #Nummer,A&,B&.......

Schreibt ein oder mehrere Byte in die Datei, die zum Datenkanal »Nummer« gehört.

PRINT #Nummer,Wert oder **String**

Schreibt einen Wert oder String in eine Datei. Der Pointer wird entsprechend erhöht. Als Abwandlung ist auch »PRINT #Nummer,USING...« zugelassen.

WRITE #Nummer,Wert oder **String**

Zwischen »PRINT« und »WRITE« bestehen folgende Unterschiede: »WRITE« verlangt, daß Strings in Anführungszeichen eingeschlossen sind. Außerdem ist es möglich, mehrere Variablen auf einmal auszugeben. Sie werden durch Kommas getrennt. Der Befehl ist darum besonders für das spätere Einlesen mit »INPUT« geeignet.

Mit folgenden Befehlen lassen sich die Daten wieder einlesen:

A&=INP(#Nummer)

Dieser Befehl erhöht den Pointer um 1 und liest dann das entsprechende Byte in die Variable »A&«.

INPUT #Nummer,Variable1,...

Liest Daten in die dem Befehl folgenden Variablen. Bei mehreren Variablen sind diese durch Kommas zu trennen. Als Variante kann auch »LINE INPUT #Nummer,Stringv1,...« verwendet werden. Beachten Sie dann bitte, daß dieser Befehl nur Strings zuläßt.

Achtung: Wie beim »normalen« INPUT-Befehl muß auch hier der Variablentyp stimmen. Wird z.B. ein numerischer Wert erwartet und statt dessen aus der Datei ein String geliefert, erfolgt der Abbruch des Programms.

A$=INPUT$(Anzahl%,#Nummer)

Liest die in der Variablen »Anzahl%« definierte Byteanzahl über den vorher geöffneten Datenkanal »Nummer« in die Variable »A$« ein. Es sind ausschließlich Strings zugelassen.

Sollen Speicherbereiche gespeichert werden, zum Beispiel ein Bildschirmbereich oder ein kompletter Bildschirm, bedient man sich sinnvollerweise der folgenden Befehle.

BPUT #Nr,Anfadr%,Anz% oder
BSAVE Pfad$,Anfadr%,Anz%

Der Unterschied zwischen beiden besteht darin, daß »BPUT« über einen vorher geöffneten Datenkanal, »BSAVE« aber über den Suchpfad speichert. »BSAVE« speichert also eine Datei vollständig, mit »BPUT« ist auch das teilweise Beschreiben einer Datei möglich. Die Variable »Anfadr%« enthält die Speicheradresse, ab der gespeichert werden soll. Die Variable »Anz%« enthält die Anzahl der zu speichernden Bytes.

Mit untenstehenden Befehlen werden Speicherblöcke eingelesen:

BGET #Nr,Anfadr%,Anz% oder
BLOAD Pfad$,Anfadr%

»BGET« lädt an die Anfangsadresse (steht in Anfadr%) die in »Anz%« vorgegebene Byteanzahl über den Datenkanal »Nr«. Durch diese Syntax können daher Teile einer Datei gelesen werden. »BLOAD« schließlich greift über einen Suchpfad auf die Datei zu und lädt sie komplett an die Anfangsadresse. Bei Verwendung von »BLOAD« kann »Anfadr%« entfallen, wenn vorher schon mit »BSAVE« gearbeitet wurde, und eine Datei an die gleiche Adresse geladen werden soll.

Die vier eben beschriebenen Befehle sind sehr schnell und sollten darum wo immer möglich eingesetzt werden.

Zwei mächtige Befehle fehlen noch. Ihnen widmen wir uns etwas ausführlicher. Mit ihrer Hilfe ist es möglich, Stringarrays ganz, teilweise oder innerhalb eines bestimmten Bereichs einzulesen oder abzuspeichern. Der Speicherbefehl heißt:

STORE #Nr,Ar$() oder
STORE #Nr,Ar$(),Anz% oder
STORE #Nr,Ar$(),Von% TO BIS%

Bei der Verwendung der ersten Syntax wird das Stringarray »AR$()« komplett über den Datenkanal »Nr« ausgegeben. Bei Verwendung der zweiten erfolgt die Ausgabe einer in »Anz%« angegebenen Menge. Die dritte schließlich läßt zu, daß sowohl der Startindex als auch der Endeindex angegeben werden können.

Beispiele:

```
STORE #1,Artikel$()
```
Es werden alle Elemente des Arrays »Artikel$« über den Kanal 1 ausgegeben.

```
STORE #1,Artikel$(),10
```
Die ersten zehn Elemente des Arrays »Artikel$« werden über den Datenkanal 1 ausgegeben.

```
STORE #1,Artikel$(),31 TO 40
```
Die Feldeinträge 31 bis 40 des Stringarrays »Artikel$« werden über den Kanal 1 ausgegeben.

Jeder Eintrag wird beim Schreiben am Ende mit ASCII 13 und ASCII 10 versehen (Carriage Return und Linefeed).

Sie können mit jedem Lesebefehl wieder eingelesen werden, am schnellsten geht's jedoch mit:

RECALL #Nr,Ar$(),Anz%,Ganz% oder
RECALL #Nr,Ar$(),Von% TO BIS%,Ganz%

Verwendet man die erste Möglichkeit, werden dem Wert der Variablen »Anz%« entsprechend viele Einträge in das vorher dimensionierte Array »Ar$« eingelesen. »Anz% = –1« transportiert die gesamte Datei ins Array. Ist die Datei kürzer als in »Anz%« angegeben, erhalten Sie keine Fehlermeldung, der Einlesevorgang wird lediglich abgebrochen. Die Variable Ganz% enthält nach dem Lesen die Anzahl der tatsächlich gelesenen Einträge.

Mit der zweiten Syntax wird ein Stringarray gezielt belegt. Die Variable »Von%« muß die Nummer des ersten Arrayelements, die Variable »Bis%« die Nummer des letzten zu belegenden Elements enthalten. Der Lesevorgang übernimmt nicht das vorher gespeicherte Carriage Return und Line Feed, beide Zeichen haben ihren Zweck als Trennzeichen erfüllt. Beispiele sind wohl überflüssig, sehen Sie bitte bei Bedarf unter »STORE« nach.

Diese Befehle beschleunigen das Speichern und Lesen ungeheuer. Konzipieren Sie darum Ihr Programm am besten so, daß Einlesen und Speichern sequentieller Datensätze über sie ablaufen.

Soviel zur Verwaltung sequentieller Dateien. Der nächste Teil behandelt den Umgang mit relativen Dateien.

5.5 Befehle für relative Dateien

Wie oben erwähnt, wird zum Öffnen einer relativen Datei als Modus "R" übergeben. Die relative Datei verlangt immer die Verwendung gleichlanger Datensätze. Jeder Datensatz darf in beliebig viele Teile geteilt werden. Die tatsächliche Länge eines Datensatzes besteht aus der Addition aller Teillängen, sie darf maximal 32767 Byte betragen.

Beispiel:
Ein Datensatz soll den Vornamen, Nachnamen und das Geburtsdatum enthalten. Vornamen sollen bis zu 15, Nachnamen bis zu 20 und das Geburtsdatum 6 Zeichen lang sein. Die Gesamtlänge eines Datensatzes beträgt daher 31 Zeichen bzw. Byte. Zum Öffnen einer entsprechenden Datei geben wir dem Rechner folgenden Befehl:

OPEN "R",#1,"NAME.DAT",31

Der Rechner öffnet eine relative Datei. Die Datenkanalnummer ist 1 und der Dateiname ist NAME.DAT. Die Datensatzlänge beträgt 31, das Ergebnis der Berechnung.

In der Eingaberoutine des gedachten Programms werden die notwendigen Angaben jeweils einer entsprechenden Stringvariablen übergeben. Ihre Namen seien: »Vname$«, »Nname$« und »Geb$«.

Das endgültige Einrichten der Datei geschieht mit:

FIELD #1,15 AS Vname$,20 AS Nname$,6 AS Geb$

Wir haben den FIELD-Befehl noch nicht beschrieben. Aus dem oben Gesagten erklärt er sich eigentlich schon von selbst. Mit »FIELD« wird jeder Datensatz von vornherein in die gewünschten Segmente unterteilt. Auf das Beispiel bezogen ergibt sich, daß die Länge der Stringvariablen »Vname$« 15 Zeichen, »Nname$« 20 Zeichen und »Geb$« 6 Zeichen betragen muß. Getrennt werden Längenangabe und Stringvariable jeweils durch »AS«. Um eine Über- oder Unterschreitung der festgelegten Länge zu vermeiden, ist es sinnvoll, den Variablen andere Inhalte ausschließlich durch: »LSET«, »RSET« oder »MID$« zuzuweisen.

Es wurde eine relative Datei geöffnet, nun wollen wir Daten speichern. Der entsprechende Befehl heißt:

PUT #Dateinr oder
PUT #Dateinr,Satznr

Bei Verwendung der ersten Syntax schreibt der Rechner den nächsten Datensatz. Die zweite schreibt den Datensatz, der in der Variablen »Satznr« übergeben wird. Dazu noch einige Erklärungen:

Bei relativen Dateien legt der Rechner intern einen Datensatzzähler an. Beim Lesen oder Schreiben wird dieser jeweils inkrementiert (hochgezählt). Wird eine Datensatznummer übergeben, muß sichergestellt sein, daß diese entweder die folgende Nummer oder daß der Datensatz schon vorhanden ist.

Für das Lesen der Datensätze sind folgende Befehle zuständig:

GET #Dateinr oder
GET #Dateinr,Satznr

Der interne Zähler kann auch voreingestellt werden. Der Befehl heißt:

RECORD #Dateinr,Satznr

Ein späterer PUT- oder GET-Befehl, ohne Angabe der Satznummer, liest dann den voreingestellten Satz.

In den meisten Fällen verwendet man beim Lesen und Schreiben eines Datensatzes wohl jeweils die zweite Syntax, und das aus folgendem Grund:

Eine relative Datei kann, solange Platz auf dem verwendeten Speichermedium zur Verfügung steht, immer vergrößert, aber nur sehr aufwendig verkleinert werden. Daß das Programm mitspielt, sei an dieser Stelle vorausgesetzt. Erinnern Sie sich bitte: Sinn und Zweck einer relativen Datei ist der wahlfreie Zugriff, d.h., man kann jederzeit über jeden vorhandenen Datensatz frei verfügen. Das Folgende setzt mindestens eine vorhandene sequentielle

Datei voraus. Jedes gut programmierte Programm läßt das Löschen einzelner Datensätze zu. Dazu ist einerseits die Berichtigung der vorhandenen Datei bzw. Dateien wichtig (darauf gehen wir nicht näher ein), andererseits muß der entsprechende Datensatz gelöscht werden. Es bieten sich zwei Lösungswege an, der schlechtere, wie wir finden, zuerst:

Man lädt die gesamte relative Datei sequentiell in den Speicher, löscht den Datensatz, d.h. entfernt ihn aus dem RAM, rückt die darauffolgenden Datensätze auf, und speichert die Datei wiederum sequentiell ab. Da die Größe einer relativen Datei nur vom Fassungsvermögen des verwendeten Speichermediums abhängig sein sollte, beginnen hier die Schwierigkeiten. Ist nur ein kleiner RAM-Bereich verfügbar, muß die Datei »zerstückelt« eingelesen und berichtigt werden. Mit anderen Worten: Je kleiner der noch verfügbare Speicher des Rechners und je größer die Datei, um so größer der Zeitaufwand beim Löschen. Es folgt die zweite Möglichkeit:

Wir berichtigen die sequentielle Datei und tragen die freigewordene Datensatznummer in einer Tabelle ein. Die Tabelle kann eine eigene Datei (bietet sich an, wenn mit mehreren sequentiellen Dateien gearbeitet wird) oder ein Teil der sequentiellen Datei bzw. Dateien sein. Die Länge der relativen Datei ändert sich dadurch nicht, die Satznummern der sequentiellen Datei bzw. Dateien deshalb ebenfalls nicht. Das einzige, was man bei der Programmerstellung zu beachten hat, ist, vor dem Speichern eines Datensatzes in der Tabelle nachzuschauen, ob ein freigegebener Satz zur Verfügung steht. Wenn ja, wird unter der entsprechenden Satznummer der Datensatz abgespeichert und die Tabelle berichtigt. Die Satznummer wird danach wie gewohnt in die sequentielle Datei eingetragen.

5.6 Dateibefehle im Einsatz

Der folgende Teil des Kapitels zeigt Ihnen einen Teil der besprochenen Befehle anhand eines Beispiels. Da wir keine Lust hatten, die 999ste Adreßdatei zu programmieren, entstand ein Programm, das sich auch bei uns im Einsatz befindet. Es handelt sich um die Verwaltung von Zeitschriften- oder Buchbeiträgen. Sie finden das Programm unter dem Namen 5_1.GFA auf der Diskette. Zunächst die Bedienung:

Nach erfolgreichem Start des Programms finden Sie eine noch »jungfräuliche« Dialogbox vor. Der Bildschirm könnte später aber wie im Bild 5.1 aussehen.

Sie haben die Möglichkeit, 18 Zeitungen und 63 Überschriften (in der Box heißen sie Themen) zu verwalten. Die Bedienung des Programms erfolgt so weit wie möglich über die Maus.

Klicken sie bitte einen Leereintrag der Zeitungsbox an. In der erscheinenden Dialogbox wird der Name der Zeitung, er darf maximal 15 Stellen lang sein, eingetragen. Nach »OK« erfolgt seine Übernahme in das vorher angeklickte Feld. Das Feld erscheint danach revers. Haben Sie keinen Namen angegeben oder »Abbruch« gewählt, bleibt der Eintrag leer, das Feld erscheint wieder »normal«. Der Zeitungsname steht parallel dazu auch noch einmal im unteren linken Teil der Box im Feld »Zeitung«.

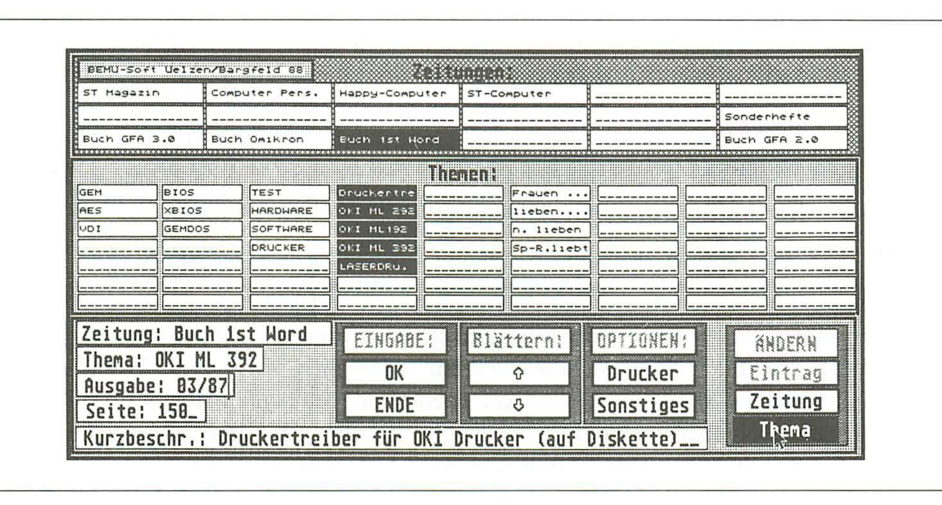


Bild 5.1: Dialogbox zur Zeitschriftenverwaltung

Der eingegebene Zeitungsname läßt sich jederzeit ändern oder löschen. Dazu klicken Sie zuerst im unteren rechten Teil der Box den Eintrag Zeitung an. Es folgt die Auswahl des zu ändernden Eintrags im Feld »Zeitungen« (er darf ruhig schon selektiert sein). Erneut erscheint die Dialogbox. Sie können den Eintrag jetzt Ihren Wünschen entsprechend ändern. Den nächsten Absatz lesen Sie bitte genau durch. Er soll verhindern, daß evtl. die Arbeit mehrerer Stunden zunichte gemacht wird.

Achtung: Wird der Eintrag gelöscht und kein neuer Name eingegeben, nimmt das Programm an, daß es auch alle Einträge, die sich auf diese Zeitung beziehen, entfernen soll. Eine Sicherheitsabfrage verhindert unbeabsichtigtes Löschen.

Zum Aufheben des Änderungs- bzw. Löschmodus wird der Eintrag »Zeitung« ein weiteres Mal angewählt. Soweit zu den Zeitungen. Kommen wir zum Themenfeld.

Das Eingeben einer Überschrift erfolgt wie bei den Zeitungen. Für das Ändern und Löschen des Namens ist der Eintrag »Thema« im unteren rechten Teil der Box zuständig, es geht dann genauso wie schon bei den Zeitungen beschrieben weiter.

Achtung: Wird beim Ändern des Namens dieser gelöscht, löscht das Programm den Namen und alle Einträge, die sich dahinter verbergen. Tun Sie sich daher bitte selbst einen Gefallen, achten Sie auf die Sicherheitsabfrage.

Hinweis: Zeitungsnamen und Überschriften (Themen) dürfen nicht zweimal vergeben werden. Das Programm achtet sehr penibel darauf und weist Sie gegebenenfalls darauf hin.

Sie haben jetzt den Namen einer Zeitung und eines Themas festgelegt, weiter geht's mit der Eingabe der zu erfassenden Daten. Hierfür dient das untere linke Feld der Box. Die Einträge »Zeitung« und »Thema« werden vom Programm selbständig ausgefüllt (wird durch

das Anklicken der entsprechenden Einträge bewirkt). Es ist immer nur eine Zeitung aktiv, Themen dürfen Sie wählen soviel Sie wollen, doch dazu später mehr. Die Einträge »Ausgabe«, »Seite« und »Kurzbeschr.« werden von Ihnen ausgefüllt. Selbst eingeben müssen Sie auf jeden Fall die Ausgabenummer der Zeitung. Sind Sie mit den Angaben zufrieden, drücken Sie die [Return]-Taste oder klicken auf »OK«. Bei Unvollständigkeit der Angaben, d.h., es fehlt der Zeitungsname, das Thema oder die Ausgabe der Zeitung, erscheint ein Hinweis. Ist alles in Ordnung, übernimmt das Programm den Eintrag.

Achtung: Überprüfen Sie bitte immer vor dem Übernehmen des Datensatzes Ihre Einträge. Ein nachträgliches Ändern ist nur mit einigem Aufwand möglich.

In einigen Fällen läßt sich ein Zeitungsartikel nicht immer einwandfrei ausschließlich einem Thema zuordnen. Das ist kein Problem, wir haben vorgesorgt. Wählen Sie die Themen aus, unter denen Sie den Artikel wiederfinden möchten. Es dürfen also auch mehrere Themen selektiert werden. Da sie aber auch nach »OK« aktiv bleiben, müssen sie, wenn gewünscht, durch erneutes Anklicken wieder deaktiviert werden. Ist mehr als ein Thema aktiv, erscheint im Feld Thema immer die zuletzt gewählte Überschrift.

Hinweis: Deaktivierung des zuletzt gewählten Themas ändert nicht den Eintrag im Feld Thema, das Programm erkennt dennoch die Änderung.

In der Box finden Sie auch den Menüpunkt »Drucker«. Wird er angewählt, druckt das Programm alle Einträge des selektierten Themas bzw. der selektierten Themen. Bei nicht eingeschaltetem Drucker erscheint ein Hinweis. Ist kein Thema aktiviert, geschieht gar nichts. Die Druckeranpassung im Programm bezieht sich auf den OKI ML 292 im IBM-Modus. Sie können sie bei Bedarf leicht ändern.

Kommen wir zum Button »Ende«. Nach Anwahl erscheint eine Dialogbox, die Ihnen drei Möglichkeiten vorstellt. Die erste ist das Sichern der Dateien und die Beendigung des Programms, die zweite das Sichern der Dateien und Rückkehr zum Programm. »Abbruch« veranlaßt das Programm, sofort zur Eingabe zurückzukehren.

Sie haben jetzt alle Funktionen kennengelernt, die dem Eingeben von Zeitungen, Themen und Einträgen und der Gesamtausgabe auf einem Drucker dienen. Das Programm kann aber selbstverständlich noch mehr. Die weiteren Befehle erreichen Sie über die Menüpunkte mit der Überschrift »Blättern«.

Hinweis: Das nun Folgende funktioniert nur, wenn mindestens ein Thema selektiert ist und mindestens ein Eintrag enthalten ist.

Sie haben also einige Einträge erfaßt und suchen nun nach einem bestimmten Beitrag. Damit Sie sich nicht jedesmal alle Einträge eines Themas ausdrucken lassen müssen, selektieren Sie einfach das entsprechende Thema bzw. die entsprechenden Themen. Jetzt klicken Sie entweder den Button mit dem Pfeil nach oben oder den mit dem Pfeil nach unten an. Nach dem Laden der entsprechenden Einträge vom aktuellen Laufwerk erscheinen im oberen Teil des Bildschirms die ersten Einträge des ersten Themas. Die Zählweise der Themen erfolgt immer von links nach rechts, die Reihenfolge beim Selek-

tieren der Themen spielt keine Rolle. Enthält ein Thema keine Einträge, werden, wenn vorhanden, die Einträge des nächsten angezeigt. Enthält das Thema mehr als vierzehn Einträge, erscheinen die nächsten durch Anklicken des Buttons mit dem Pfeil nach unten. Damit Sie nicht den Anschluß verlieren, erscheint der letzte Eintrag der vorherigen Seite jetzt als erster. Zurückgeblättert wird mit dem Button, der den Pfeil nach oben enthält. Die letzte Zeile der Seite zeigt dann den Inhalt der vormals ersten.

Sind mehrere Themen selektiert, werden Sie schnell feststellen, daß Sie sich beim Blättern immer nur innerhalb eines Themas befinden. Das ist so gewollt! Zum Anzeigen der Einträge des nächsten Themas klicken Sie bitte zuerst »OK« an. Danach bewirkt das Anklicken des Pfeils nach unten das Laden der Einträge des nächsten Themas. Anklicken des Pfeils nach oben lädt die Einträge des vorherigen Themas. Das Ganze funktioniert selbstverständlich nur, wenn es ein nächstes oder vorheriges Thema mit Einträgen gibt. Haben Sie schon bemerkt, daß sich der erste Eintrag der unteren rechten Box von »Ändern« in »Löschen« gewandelt hat? Lesen Sie dazu den nächsten Absatz.

In dieser Box ist nur noch der Button »Eintrag« wählbar. Die ganze Kombination heißt also: »Lösche Eintrag«. Sie sind bestimmt schon selbst drauf gekommen. Vor dem Löschen eines Eintrags, müssen wir dem Programm noch mitteilen, um welchen es sich handelt. Das Auswählen geschieht, wie soll es anders sein, durch Anklicken des Eintrags. Damit sind wir beim Selektieren von Einträgen angekommen.

Jeder Eintrag auf dem Bildschirm läßt sich durch Anklicken selektieren. Er erscheint danach revers. Haben Sie sich geirrt, reicht ein nochmaliges Anklicken, und der Eintrag wird wieder normal dargestellt. Sie dürfen beim Vor- und Zurückblättern so viele Einträge aktivieren, wie Sie möchten. Das Programm »merkt« sich alle Einträge und stellt sie entsprechend dar. Es gibt jedoch eine Einschränkung:

Ein Aufruf des vor- oder zurückliegenden Themas durch »OK« löscht das Selektiergedächtnis des Programms. Mit selektierten Einträgen läßt sich folgendes machen: Sie können diese Einträge vom Drucker drucken lassen oder löschen. Beides zusammen geht auch.

Sollen Einträge gelöscht werden, empfehlen wir Ihnen, wie folgt vorzugehen:

- Selektieren Sie die zu löschenden Einträge des Themas.
- Schalten Sie den Drucker ein, und klicken Sie danach auf Drucker.
- Nach dem Drucken wählen Sie den Button Eintrag.
- Die Sicherheitsabfrage zum Löschen beantworten Sie positiv.

Halten Sie diese Reihenfolge ein, erhalten Sie einen Überblick über die unwiederbringlich gelöschten Einträge. Irrtümlich gelöschte Einträge sind anhand der Druckerausgabe schnell wieder eingegeben. Noch ein Hinweis:

Sie haben einen Datensatz namens »Eintrag 1« bei der Eingabe den Themen »Eins«, »Zwei« und »Drei« zugeordnet, und scrollen durch Thema »Eins«. Wird der Eintrag »Eintrag 1« hier gelöscht, bleibt er weiterhin in den Themen »Zwei« und »Drei« erhalten.

Sie erhalten damit eine Möglichkeit, die Zuordnung in einem begrenzten Rahmen noch zu verändern.

Haben Sie alle Informationen bekommen bzw. sind Sie mit der Dateipflege fertig, so führt Sie der Button »Ende« wieder zur Ausgangsbox zurück.

Einen Menüpunkt haben wir noch nicht angesprochen, nämlich »Sonstiges«. Er dient zum Erweitern des Programms. Ideen dazu werden Sie bestimmt haben.

Mit Hilfe der vorstehenden Anleitung können Sie das Programm schon mal bedienen. Viele Leser möchten es bestimmt jetzt oder später ändern bzw. erweitern. Darum folgt zunächst eine allgemeine Beschreibung des Programmablaufs.

Zuerst werden im Unterprogramm Definitionen wichtige Variablen mit Werten vorbelegt und später verwendete Felder dimensioniert. Dabei handelt es sich zum Beispiel um die Suchpfade der einzelnen Dateien und Druckersteuerzeichen, außerdem wird Speicher für die RSC-Datei reserviert. Nach der Abarbeitung lädt das Programm die Resource-Datei ein, das Programm läuft ja fast vollständig unter GEM. Deshalb existiert auch eine Prozedur namens »Objekt_nr«. In ihr werden die einzelnen Objektnummern, und das sind nicht wenige, entsprechenden Variablen zugewiesen. Nach diesen Vorbereitungen kann die Dialogbox vom GEM gezeichnet werden, das erledigt der Aufruf des Unterprogramms »Dialogbox_zeichnen«. Das Unterprogramm »Box_abfrage« fragt danach die Benutzeraktivitäten ab und reagiert, mit Hilfe weiterer Routinen, auf sie. Im Folgenden wollen wir Ihnen die Routinen, die sich mit der Dateiverwaltung beschäftigen, etwas ausführlicher erklären.

Das Programm erstellt drei Dateien. Sie heißen: ZEI_T.DAT, ZEI_Z.DAT und ZEI_R.DAT. Die ersten beiden Dateien sind sequentielle Dateien, bei der Datei ZEI_R.DAT handelt es sich um eine relative Datei. Die Datei ZEI_Z.DAT ist am schnellsten beschrieben, sie enthält die 18 möglichen Zeitungsnamen. Die Datei ZEI_T.DAT enthält die Nummern der gelöschten Datensätze, die Namen der Themen und die Datensatznummern, unter denen Einträge zu finden sind. Die sequentiellen Dateien werden beim Laden in vorher dimensionierte Stringarrays eingelesen. Die relative Datei schließlich enthält eine Zugriffsnummer auf die Zeitung, die Angabe der Seite, die Ausgabe der Zeitung und den Text der Kurzbeschreibung.

Bevor mit Dateien gearbeitet werden kann, müssen sie erst einmal eingelesen werden, eine Aufgabe, die die untenstehende Routine erledigt.

```
PROCEDURE dateien_laden
  LOCAL i&,i1&
  IF EXIST(dat_thname$)
    t$(0)=SPACE$(130)
    OPEN "I",#1,dat_thname$
    BGET #1,VARPTR(t$(0)),130
    i1&=1
    FOR i&=1 TO 65
```

```
      lae&=CVI(MID$(t$(0),i1&,2))
      t$(i&)=SPACE$(lae&)
      BGET #1,VARPTR(t$(i&)),lae&
      ADD i1&,2
    NEXT i&
    CLOSE #1
  ENDIF
```

Der obenstehende Unterprogrammteil liest die Themen in das Array »t$« ein. Zuerst überprüft das Programm, ob überhaupt ein File, dessen Suchpfad und Namen in »dat_thname$« stehen, auf der Diskette vorhanden ist. Bei Vorhandensein wird der Array-Eintrag 0 mit 130 Leerzeichen gefüllt. Bevor wir weitermachen, zunächst die Belegung des Arrays »t$«:

In »t$(0)« stehen die Längen der Feldelemente 1 bis 65. Die Elemente 1 bis 63 haben folgenden Aufbau: Die ersten zehn Byte repräsentieren den Namen des entsprechenden Themas, die ihm folgenden Bytes die Datensatznummern, unter denen über dieses Thema etwas zu finden ist. Jede Datensatznummer besteht aus zwei Byte. Im Feldeintrag »t$(64)« stehen die durch einen Löschvorgang freigewordenen Datensatznummern.

Eine besondere Aufgabe fällt dem Eintrag »t$(65)« zu. Wie Sie schon der Bedienungsanleitung entnehmen konnten, ist es möglich, Einträge zu löschen. Das Programm läßt bei der Erfassung der Daten auch die Wahl mehrerer Themen für einen Eintrag zu. Wir hatten die Wahl:

Entweder Freigabe des Datensatzes beim Löschen, d.h., er erscheint auch bei der Auflistung anderer Themen, unter denen er evtl. einmal abgelegt worden ist, nicht mehr, oder man löscht ihn tatsächlich nur aus dem gewählten Thema. Wir haben uns für letztere Methode entschieden, mußten nun aber noch etwas bedenken: Angenommen, auf den dritten Datensatz haben die Themen 1 und 2 Zugriff. Zuerst löscht jetzt der Anwender den Zugriff im Thema 1. Das Programm muß nun feststellen, ob auch noch andere Themen Zugriff haben. Wenn ja, bleibt der Datensatz bestehen, wenn nein, wird er freigegeben, sprich, die Nummer in »t$(64)« eingetragen. Löschen des Eintrags bei Thema 2 erzwingt die gleiche Vorgehensweise. Es ist schnell erkennbar, daß die Überprüfung um so länger dauert, je mehr Datensätze vorhanden sind. Wir gingen daher einen etwas anderen Weg, und damit sind wir wieder beim Feldeintrag »t$(65)«. Jedes Byte dieses Strings repräsentiert einen Zähler für jeden vergebenen Datensatz. Byte 1 ist der Zähler für Datensatz 1, Byte 2 der Zähler für den Datensatz mit der Nummer 2 usw. In das entsprechende Byte trägt das Programm beim Speichern eines Datensatzes ein, wie viele Themen auf ihn Bezug nehmen. Beim Löschen eines Eintrages braucht nur dieses Byte dekrementiert zu werden. Die darauffolgende Prüfung auf 0 entscheidet über die Freigabe der Datensatznummer. Zurück zum Einlesen des Themenfeldes.

Das Array läßt sich leider nur etwas umständlich einlesen. Das rührt daher, daß der Befehl »RECALL« beim Einlesen die Trennung der einzelnen Elemente anhand von ASCII 13 und ASCII 10 vornimmt. Bei unserer Version des Basic wird sogar nur nach ASCII 13 gesucht und dann getrennt. Wie schon gesagt, stehen in der Themendatei die Datensatznummern.

Trifft GFA beim Einlesen auf die Nummer 13, faßt es diese als Trennzeichen auf und beendet für das entsprechende Feldelement den Einlesevorgang. Mit anderen Worten, es gerät alles durcheinander!

Nach dem Öffnen der Datei liest das Programm die ersten 130 Byte mit »BGET« an die Adresse von »t$(0)«. In der folgenden Schleife werden danach, entsprechend der Längenangabe von »t$(0)«, die anderen Feldelemente (1 bis 65) nach dem gleichen Verfahren in das Feld gelesen. Zum Schluß schließt das Programm den geöffneten Datenkanal.

```
  IF EXIST(dat_zname$)
    OPEN "I",#2,dat_zname$
    RECALL #2,z$(),-1,az%
    CLOSE #2
  ENDIF
```

Die Zeitungsnamen lassen sich wesentlich schneller als die Themen einlesen. Das haben wir ausgenutzt und darum den Befehl »RECALL« eingesetzt.

Zum Schluß wird noch die relative Datei geöffnet.

```
  OPEN "R",#3,dat_dsname$,56
  l_d%=LOF(#3)
  IF l_d%>0
    lds&=l_d%/56                      !Letzter Datensatz
  ENDIF
  FIELD #3,1 AS zeitungs_nr$,4 AS seite$,4 AS ausgabe$,47 AS kurzb$
RETURN
```

Nach dem Öffnen stellen wir die Länge der Datei mit »LOF« fest. Aus ihr errechnet das Programm die letzte Datensatznummer. Sie steht danach in der Variablen »lds&«. Der Befehl »FIELD« teilt schließlich jeden Datensatz in verschiedene Segmente auf.

Jeder Datensatz hat eine Länge von 56 Byte. Die Belegung der Bytes ist wie folgt: Das erste Byte enthält die Nummer der Zeitung, wir sparen dadurch Speicherplatz auf der Diskette. Rechnen wir einmal: Jeder Zeitungsname darf 15 Zeichen lang sein. Bei 100 Datensätzen beträgt der notwendige Speicherbedarf allein für die Namen in der relativen Datei 1500 Byte. Bei der vorliegenden Dateienorganisation benötigen wir in der sequentiellen Datei 15 Byte pro Namen und jeweils 2 Byte als Trennzeichen zwischen ihnen. Das Ganze mal 18 ergibt 306 Byte. Hinzu kommen nur noch die 100 Byte, die wir in der relativen Datei für die Zeitungsnummer brauchen. Wir sparen in diesem Fall bereits 1094 Byte. Je mehr die relative Datei wächst, um so größer ist die Speicherersparnis. Der Rest der Aufteilung ist fix erklärt. Die Einträge der Seite und der Ausgabe belegen je 4, die Kurzbeschreibung 47 Byte.

Zum Sichern eines neuen Datensatzes ruft das Programm die Prozedur »datensatz_sichern« auf.

```
PROCEDURE datensatz_sichern
  '
  LOCAL i&,i1&
  IF LEN(t$(64))=0
    INC lds&                          !letzte Datensatznummer
    ds_nr&=lds&                       !Datensatznummer
    t$(65)=t$(65)+CHR$(LEN(th$))
  ELSE
    ds_nr&=CVI(LEFT$(t$(64),2))
    t$(64)=RIGHT$(t$(64),LEN(t$(64))-2)
    MID$(t$(65),ds_nr&,1)=CHR$(LEN(th$))
  ENDIF
```

Zuerst erfolgt die Überprüfung auf freigegebene Datensatznummern. Ist die Länge von »t$(64)« gleich 0, berechnet das Programm die neue zu vergebende Satznummer. Bei einer Länge von »t$(64)« größer als 0, besorgt sich das Programm eine Nummer aus diesem Feldeintrag. Danach wird sie aus ihm entfernt, der Datensatz ist ja nicht mehr frei. In »t$(65)« trägt das Programm jeweils die Anzahl der Themen ein, unter denen der Eintrag zu finden ist.

```
  FOR i&=1 TO LEN(th$)
    i1&=ASC(MID$(th$,i&))
    t$(i1&)=t$(i1&)+MKI$(ds_nr&)
  NEXT i&
  PUT #3,ds_nr&
RETURN
```

In der Variablen »th$« stehen die Indizes der Themen. In der Schleife wird die ermittelte Datensatznummer den gewählten Themen zugeordnet. Schließlich sichert das Programm den Datensatz mit dem PUT-Befehl. Das nächste Unterprogramm löscht eine Zeitung vollständig. Es weist einige Besonderheiten auf, die wir zuerst erklären möchten. Wir haben schon erwähnt, daß sich eine relative Datei auch sequentiell öffnen läßt. Das kann dann sinnvoll sein, wenn lediglich Teile eines Datensatzes gebraucht werden. Das ist bei uns der Fall, wir benötigen nur die Zeitungsnummer.

```
PROCEDURE zeitung_loeschen(zi&)
  '
  ' zi&=> Nummer der zu löschenden Zeitung
  LOCAL i&
  z$(zi&)=""
```

Zuerst löscht das Programm den Zeitungsnamen

```
 OPEN "U",#1,dat_dsname$
```

Die relative Datei wird ein weiteres Mal geöffnet, jetzt aber als sequentielle Datei.

```
  IF LOF(#1)>0
    FOR i&=0 TO lds&-1
      SEEK #1,i&*56
      IF INP(#1)=zi&
        RELSEEK #1,-1
        OUT #1,0
      ENDIF
    NEXT i&
  ENDIF
  CLOSE #1
RETURN
```

Damit es keine Fehlermeldung gibt, muß zunächst überprüft werden, ob überhaupt schon Datensätze in der Datei stehen. In der Schleife wird der Dateipointer nacheinander immer auf das erste Byte jedes Datensatzes positioniert (Befehl SEEK) und dieses Byte über »INP« gelesen. Stimmt die Zeitungsnummer mit der zu löschenden überein, erfolgt das Rücksetzen des Datenzeigers auf eben die gelesene Position (Befehl RELSEEK) und anschließendes Schreiben der Zahl 0. Nach beendeter Arbeit schließt das Unterprogramm die sequentielle Datei wieder. An dieser Stelle wurde also noch kein Datensatz gelöscht, sondern nur die Sätze markiert, die sich auf die zu löschende Zeitung beziehen. Das endgültige Löschen bzw. Freigeben der markierten Datensätze übernimmt eine andere Routine. Wir wählten diesen Weg aus folgendem Grund:

Angenommen, es existieren 2000 Datensätze und 40 Themen. Um eine Zeitung in einem Durchgang zu löschen, müßte man die Zeitungsnummer des ersten Datensatzes lesen. Ist sie mit der zu löschenden identisch, hat das Programm die 40 Themen auf Existenz der Datensatznummer 1 zu überprüfen und, wenn vorhanden, zu löschen. Selbst wenn sich nur 200 Datensätze auf die Zeitung beziehen, ist der eben beschriebene Vorgang bereits 8000mal erforderlich. Das bedeutet auch unter GFA-Basic einen recht hohen Zeitaufwand. Das Freigeben der Datensätze übernimmt die folgende Routine. Sie wird immer aufgerufen, wenn Datensätze eingelesen werden sollen.

```
PROCEDURE daten_lesen(th_in&,VAR max&)
  '
  ' Diese Procedure liest Datensätze ein. Ist die Zeitungsnr.=0, d.h.,
  ' Zeitung wurde gelöscht, wird der Datensatz aus dem Themastring
  ' entfernt.
  '
  ' th_in&=>Thema Index
  ' z&=> Zeiger
  ' a_ds&=> Anzahl der Datensätze (Zeilen)
  '
  LOCAL ds_nr&,z&
  CLR a_ds&,sel$
  scroll_ue$=LEFT$(t$(th_in&),10)
```

```
max&=(LEN(t$(th_in&))-11)/2+1
ERASE p$()
DIM p$(max&)
z&=11
```

Nach einigen Vordefinitionen bildet das Programm zuerst die Scrollüberschrift. Für die Scrollroutine benötigen wir die maximal mögliche Anzahl der Zeilen. Sie läßt sich aus der Anzahl der Datensatznummern des entsprechenden Themenfeldeintrags errechnen. In das Stringarray »p$« werden später die Datensätze gelesen. Das Feld wird dann auf dem Bildschirm gescrollt. Vor dem Lesevorgang wird das Array zuerst entfernt und dann neu angelegt (Befehle ERASE und DIM).

```
WHILE z& LEN(t$(th_in&))
  INC a_ds&
  ds_nr&=CVI(MID$(t$(th_in&),z&,2))
  GET #3,ds_nr&
```

Diese Schleife liest die entsprechenden Datensätze nacheinander ein. Nach jedem Ladevorgang erfolgt die Prüfung auf die Zeitungsnummer 0.

```
    IF ASC(zeitungs_nr$)=0
      DEC a_ds&
      DEC max&
      t$(th_in&)=LEFT$(t$(th_in&),z&-1)+RIGHT$
        (t$(th_in&),LEN(t$(th_in&))-(z&+1))
      SUB z&,2
      IF ASC(MID$(t$(65),ds_nr&,1))=1
        t$(64)=t$(64)+MKI$(ds_nr&)
      ELSE IF ASC(MID$(t$(65),ds_nr&,1))>0
        MID$(t$(65),ds_nr&,1)=CHR$(ASC(MID$(t$(65),ds_nr&,1))-1)
      ENDIF
```

Handelt es sich bei dem eingelesenen Datensatz um einen markierten, wird zunächst die Datensatznummer aus dem Themenstring entfernt. Bezieht sich auf den Datensatz nur noch ein Thema, erfolgt die Freigabe der Datensatznummer. Sie kann dann später erneut vergeben werden. Im anderen Fall wird der Zähler lediglich dekrementiert (heruntergezählt).

```
    ELSE
      p$(a_ds&)=z$(ASC(zeitungs_nr$))
      p$(a_ds&)=p$(a_ds&)+"  "+LEFT$ (ausgabe$,2)+"/"+RIGHT$(ausgabe$,2)
      p$(a_ds&)=p$(a_ds&)+"   "+seite$+"   "+kurzb$
    ENDIF
    ADD z&,2
  WEND
RETURN
```

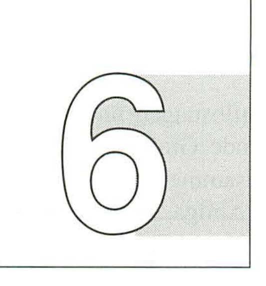
Ist die Zeitungsnummer größer als 0, steht einer Übernahme in das Scrollarray nichts mehr im Wege. Die Übernahme erfolgt mit obigen Programmzeilen. Nach getaner Arbeit müssen die veränderten sequentiellen Dateien gesichert werden. Diese Aufgabe übernimmt die Prozedur »alles_sichern«.

```
PROCEDURE alles_sichern
  '
  ' Sichert die sequentiellen Dateien
  '
  LOCAL i&,bu&
  t$(0)=""
  ALERT 2,"SICHERN UND WEITER oder| |SICHERN UND
           ENDE?",1,"SICHERN|S.& ENDE |ABBRUCH",bu&
  IF bu& 3
    OPEN "O",#1,dat_thname$
    OPEN "O",#2,dat_zname$
    FOR i&=1 TO 65
      t$(0)=t$(0)+MKI$(LEN(t$(i&)))
    NEXT i&
    FOR i&=0 TO 65
      BPUT #1,VARPTR(t$(i&)),LEN(t$(i&))
    NEXT i&
    CLOSE #1
    STORE #2,z$()
    CLOSE #2
  ENDIF
```

Hat sich der Anwender für das Sichern entschieden, öffnet das Programm zunächst zwei sequentielle Dateien. Dabei verwendet es selbstverständlich wieder dieselben Dateinamen wie beim Einlesen. Als nächster Schritt liegt das Feststellen der Längen der Einträge »t$(1-65)« an. Sie werden in »t$(0)« festgehalten. In der Schleife speichert das Programm dann mit »BPUT« das Themenfeld ab. Das Feld mit den Zeitungsnamen kann mit der wesentlich schnelleren Funktion »STORE« gesichert werden.

```
  ~OBJC_CHANGE(adr%(1),exbu&,0,x&(1),y&(1),b&(1),h&(1),0,1)
  IF bu&=2
    GOSUB ende
  ENDIF
RETURN
```

Die letzten Zeilen des Unterprogramms erklären sich wohl von selbst, wir verzichten daher auf eine Beschreibung. Das war's zum Thema Dateiverwaltung. Wenn Sie jetzt am Ende des Kapitels gut gelaunt und voller Elan an die Programmierung Ihrer eigenen Dateiverwaltung gehen, hat sich das Lesen dieses wirklich trockenen Themengebietes schon gelohnt.



6 Grafik

Moderne Computer, wie der Atari ST, sind selbstverständlich in der Lage, hochauflösende Farbgrafik darzustellen. Noch vor wenigen Jahren war dieses faszinierende Anwendungsgebiet für Computer auf Spezialgeräte beschränkt. Der »breiten Masse« der Anwender standen mit den einfachen Heimcomputern nur primitive Werkzeuge für die Grafikprogrammierung zur Verfügung. Durch äußerst geschickte Programmierung sind aber selbst auf den kleinen Rechnern beachtliche Leistungen möglich.

Mittlerweile werden preiswerte Computer angekündigt, die Grafikauflösungen versprechen, die alles bisher Dagewesene in den Schatten stellen. Wir mit unserem Atari ST stehen sozusagen zwischen diesen Computerwelten. Der ST kann eine sehr hoch auflösende Grafik mit 640 x 400 Bildschirmpunkten darstellen, jedoch mit nur zwei »Farben«, nämlich Schwarz und Weiß. Wer mehr Farben haben möchte, muß auf Teile der Auflösung verzichten. So sind im Modus mit vier Farben nur 640 x 200 Punkte möglich, Im 16-Farben-Modus gar nur 320 x 200 Punkte.

Die Programme in diesem Buch machen hauptsächlich von der höchsten Grafikauflösung Gebrauch. Die meisten ST-Besitzer werden sicher auch im Besitz des Monitors SM 124 sein. Das ist bekanntlich der sehr gute und augenschonende Schwarzweiß-Monitor von Atari. Die meisten modernen Anwendungen für den Computer, zum Beispiel *Desktop Publishing*, verlangen geradezu diesen Monitor. Nun ist so etwas sicher auch in Farbe sehr schön, das Problem besteht jedoch darin, die Grafiken auch aufs Papier zu bekommen. Ernsthafte Anwendungen benötigen hierfür die Laserdrucker, und diese drucken bisher leider nur einfarbig. Die Farbdrucker, Matrixdrucker mit verschiedenfarbigen Farbbändern, sind für die ernsthaften Anwendungen weniger gut geeignet. Andere Anwendungen wie CAD (*Computer Aided Design* = computerunterstützter Entwurf), CAM (Computer Aided Manufacturing = computerunterstütztes Fertigen) usw. sind weniger auf Farben, dafür jedoch mehr auf eine hohe Auflösung angewiesen. Dagegen benötigt man z.B. im künstlerischen Bereich oder für Computeranimation in der Regel eine reichhaltige Farbpalette.

6.1 Der Bildschirmausgabeteil des Atari ST

Betrachten wir zunächst einmal die verschiedenen Bildschirmauflösungen und die Verwaltung des Bildschirmspeichers des Atari ST. Für hochauflösende Grafik ist eine ausgeklügelte Hardware des Rechners unverzichtbar. Diese bildet zusammen mit der starken Software die heutigen modernen Grafiksysteme wie Atari ST oder Amiga. In unserem Atari ST ist hierfür dank der steten Weiterentwicklung der hochintegrierten Chips im wesentlichen nur noch ein einziger Baustein eingesetzt: der sogenannte »Shifter«. Dieser sorgt dafür, daß ein Videosignal erzeugt wird, das alle Informationen für die Farbgrafik enthält. Es wird hierzu ein spezieller Speicherbereich (Video-RAM oder »Bildschirmspeicher«) genutzt. Hier sind die Grafikdaten in Form von Farbnummern gespeichert. Der Shifter liest nun mit Hilfe einiger weiterer Chips diesen Speicherbereich aus und bildet das schon angesprochene Videosignal, das die Bildwiedergabe auf Ihrem Schwarzweiß- oder Farbmonitor ermöglicht.

Der Shifter ist darüber hinaus noch in der Lage, verschiedene Grafikmodi zu unterstützen.

- Hohe Auflösung (640 x 400 Punkte).
  Je Punkt 1 Bit Farbinformationen (2 Farben).
- Mittlere Auflösung (640 x 200 Punkte).
  Je Punkt 2 Bit Farbinformationen (4 Farben).
- Niedrige Auflösung (320 x 200 Punkte).
  Je Punkt 4 Bit Farbinformationen (16 Farben).

Verwechseln Sie diese Darstellungsmodi nicht mit dem Befehl »GRAPHMODE«, das ist nämlich etwas ganz anderes. Wir werden später noch darauf zurückkommen.

Der Bildschirmspeicher, der für die Speicherung der Grafikdaten verwendet wird, ist stets 32.000 Byte groß. Sie brauchen sich um diesen Speicherbereich keinerlei Gedanken zu machen, da der Rechner selbst die Verwaltung übernimmt. Trotzdem ist es natürlich manchmal notwendig, zum Beispiel die Anfangsadresse des Bildschirmspeichers zu erfahren. Auch das ist recht einfach durch XBIOS-Funktionen möglich.

Adresse_Phys = XBIOS(2)
Adresse_Log = XBIOS(3)

Hier sehen Sie gleich eine Besonderheit dieses Rechners. Für den Bildschirmspeicher gibt es nämlich zwei Adressen, die *physikalische* und die *logische* Anfangsadresse. Der physikalische Bildschirmspeicher ist der Speicherbereich, der auch auf dem Bildschirm stets sichtbar ist, während die Adresse des logischen Bildschirmspeichers auf den Bereich zeigt, der von den Grafikbefehlen genutzt wird. Sinn dieser ganzen Prozedur ist der, in einem Bildschirmspeicher mittels Grafikbefehlen ein Bild zu erstellen, während ein anderes Bild angezeigt wird. So etwas wird durchaus häufig angewandt, zum Beispiel, um Zeit zu sparen oder dem Anwender nicht die manchmal etwas langsame Erstellung eines Bildes, sondern erst die komplette Grafik zu zeigen. An anderer Stelle dieses Buches (Kapitel 7) zeigen wir

Ihnen, wie die physikalische und die logische Bildschirmadresse beeinflußt werden kann und welche Möglichkeiten sich dadurch für Sie ergeben.

Nomalerweise zeigen beide angesprochenen Adressen auf denselben Speicherbereich, was bedeutet, daß alle Grafikausgaben auf den Bildschirm gehen, der auch im Moment sichtbar ist. Nun haben wir von den verschiedenen Grafikmodi berichtet und davon, daß der Bildschirmspeicher immer 32.000 Byte groß ist. Die folgenden drei Bilder sollen Ihnen zeigen, wie die verschiedenen Modi im Speicher Ihres Rechners »aussehen«. Sie zeigen den Zusammenhang zwischen Bildschirmspeicher und Darstellung auf dem Monitor.

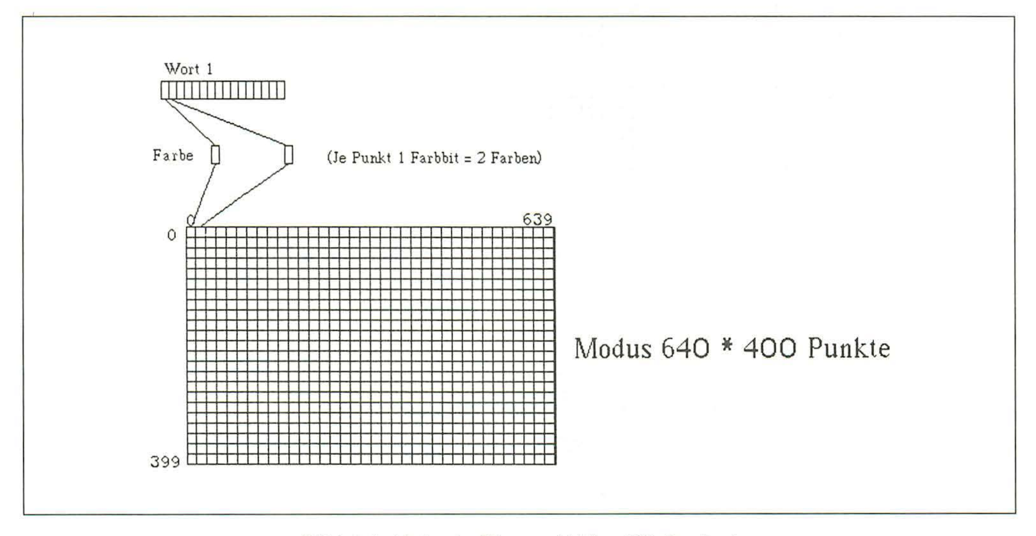


Bild 6.1: Hohe Auflösung (640 x 400 Punkte)

Wir haben in den drei Bildern jeweils für die ersten zwei Bildpunkte (auf dem Bildschirm oben links) die Zusammenhänge zwischen Bildschirmspeicher und Farbbestimmung dargestellt. Somit ist erkennbar, daß bei der hohen Auflösung jedes einzelne der 16.000 Wörter des Bildschirmspeichers für jeweils 16 Bildpunkte zuständig ist (jedes Bit kann die zwei möglichen Farben darstellen). Bei der mittleren Auflösung enthalten jeweils zwei Wörter die Informationen für 16 Bildpunkte (je Punkt 2 Bit Farbinformationen). In der dritten Auflösung, der niedrigen Bildschirmauflösung, werden die 16 Bildpunkte durch vier aufeinanderfolgende Wörter definiert (je Punkt 4 Bit Farbinformationen). Man spricht dann von vier *Planes*.

Bei allen Programmen, die direkt auf den Bildschirmspeicher zugreifen, muß also auf diese sehr unterschiedlichen Speicherorganisationen geachtet werden.

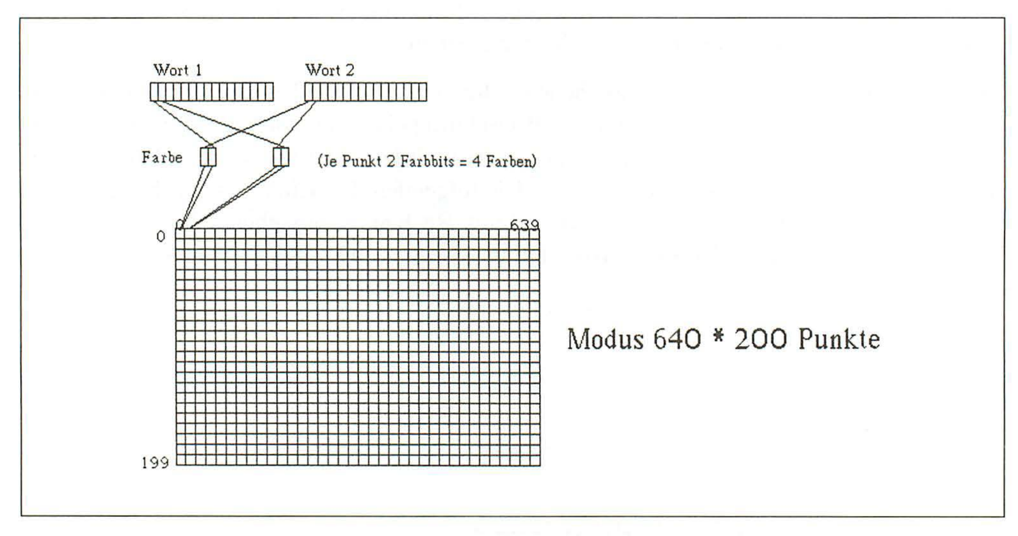


Bild 6.2: Mittlere Auflösung (640 x 200 Punkte)

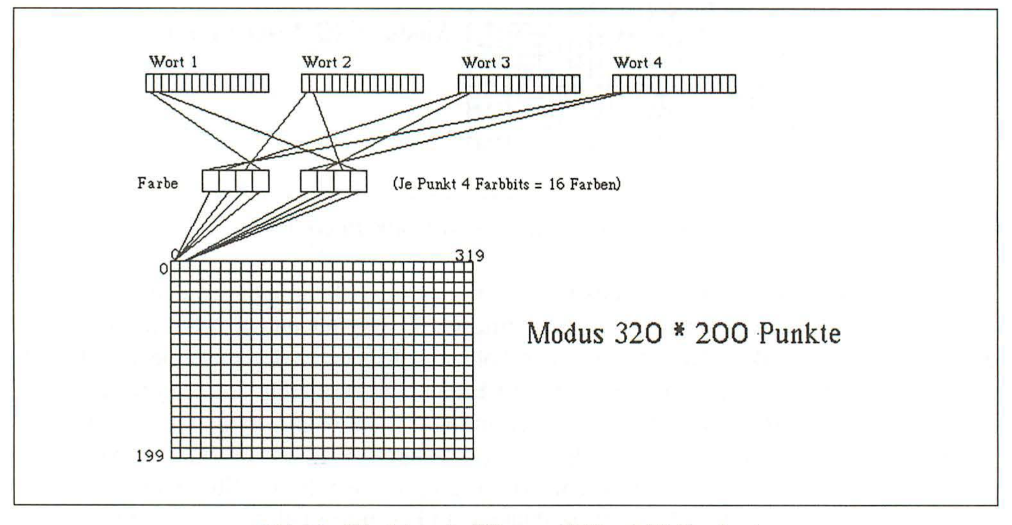


Bild 6.3: Niedrige Auflösung (320 x 200 Punkte)

6.2 Farbgrafik

Die im Bildschirmspeicher gesetzten Bits (je nach Auflösung 1, 2 oder 4 Bit) geben eine Farbnummer an.

Im vorliegenden Buch behandeln wir hauptsächlich die hochauflösende Grafik (640 x 400 Punkte). Hierbei besteht die »große Auswahl« zwischen den Farben nur noch aus Schwarz und Weiß. Deshalb können und möchten wir aber trotzdem nicht ganz die Befehle des GFA-Basic übergehen, die für die Farbbestimmung vorgesehen sind. Vorweg noch die Anmerkung, daß der Atari ST zweifellos herrliche Farbgrafiken zaubern kann, die aber auf Kosten der Auflösung zustande kommen. Außerdem gibt es keine preiswerten Farbmonitore, die auch nur annähernd an die hohe Qualität des (Schwarzweiß-)Monitors SM124 von Atari herankommen. Sie haben also bei der Farbgrafik nicht nur eine schlechtere Auflösung, sondern darüber hinaus sowieso ein schlechteres Bild. Das macht sich dann mitunter sogar mit Kopfschmerzen bemerkbar. Nun kann man die Farbgrafik natürlich nicht gänzlich verdammen, es gibt nämlich gute Farbgrafikprogramme und Spiele für den ST, die die Farbmöglichkeiten bestens ausnutzen. Wer sich außerdem mit dem Digitalisieren irgendwelcher Bilder beschäftigt, kommt auch früher oder später auf die Farbgrafik. Der Atari ist auch in der Farbgrafik noch leistungsfähiger als manch anderer Computer. Gerade im künstlerischen Bereich oder bei den Spielen kommt man kaum um die Farbgrafik herum.

Wenn Sie einen Farbmonitor betreiben, ist nach Einschalten des Systems die niedrige Bildschirmauflösung (320 x 200 Punkte mit 16 Farben) eingestellt. Die Auflösung können Sie dann nur noch durch die Option »Voreinstellungen«, erreichbar im Desktop durch das Pulldown-Menü »Extras« der durch eine XBIOS-Funktion aus dem Basic heraus ändern. Mit dem Befehl

```
XBIOS(5,L:log%,L:phys%,aufl%)
```

werden die Adressen für den physikalischen und den logischen Bildschirmspeicher neu festgelegt. Außerdem kann die Bildschirmauflösung angegeben werden. Sie können dann also zum Beispiel von der niedrigen zur mittleren Auflösung umschalten. Dann werden Sie übrigens auch seltsame Effekte feststellen, wenn Sie mit »Quit« zurück auf das Desktop gehen. Dort ist nämlich dann mitunter nichts mehr zu erkennen. Wird der Schwarzweiß-Monitor verwendet, können Sie nicht in die mittlere oder niedrige Auflösung umschalten. Der Monitor kann ja bekanntlich keine Grauwerte darstellen. Ein Versuch, das doch zu erreichen, wird mit einem Reset geahndet.

Wenn Sie aus einem Programm heraus die Bildschirmauflösung feststellen möchten, können Sie das ebenfalls mit einer XBIOS-Funktion machen:

Aufl%=XBIOS(4)

So, die Anzahl der Farben ist jetzt definiert. Nun müssen diese Farben selbst nur noch festgelegt werden und lassen sich dann auf dem Bildschirm darstellen. Folgende Befehle stehen zur Verfügung:

SETCOLOR Reg,Rot,Gruen,Blau oder
VSETCOLOR Reg,Rot,Gruen,Blau

bestimmt die Farbe, bestehend aus den drei Grundfarben Rot, Grün und Blau (jeweils Werte von 0 bis 7), für das angegebene Farbregister (0 bis 15). Der Unterschied zwischen »SETCOLOR« und »VSETCOLOR« ist, daß die einzelnen Farbregister unterschiedlich zugeordnet sind. Die folgende Tabelle soll dies verdeutlichen.

SETCOLOR	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
entspricht																
VSETCOLOR	0	2	3	6	4	7	5	8	9	10	11	14	12	15	13	1

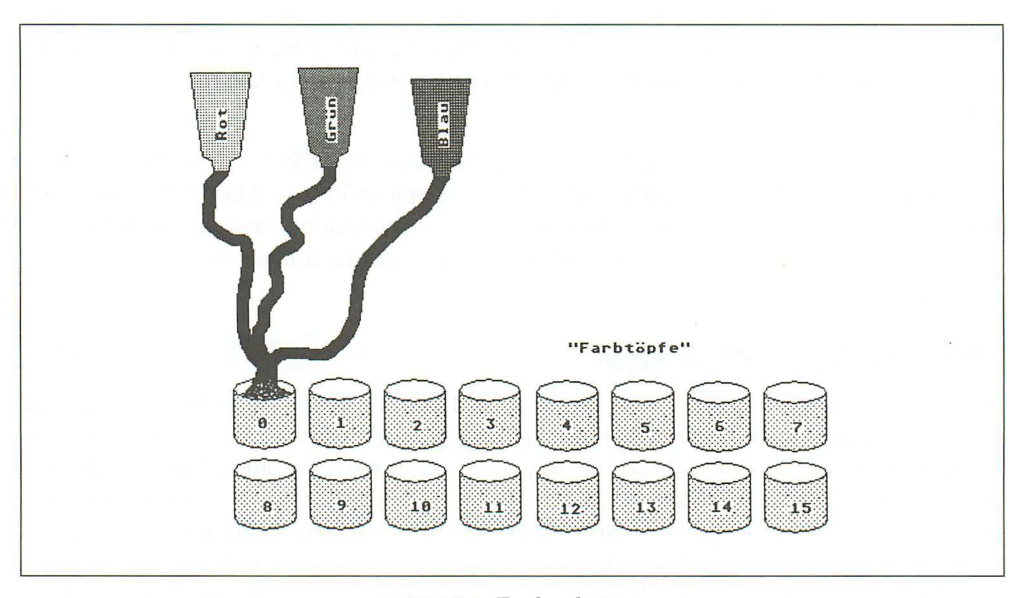


Bild 6.4: Farbpalette

Statt der einzelnen Werte der Grundfarben kann auch ein kombinierter Wert angegeben werden. Dieser setzt sich aus

rot x 256 + gruen x 16 + blau

zusammen. Vorteilhafterweise macht man jedoch die Angabe als Hexadezimalwert, wie das folgende Beispiel zeigt:

```
SETCOLOR 4,&H722
```

Die jeweils acht möglichen Werte der Grundfarben geben natürlich die Helligkeit der einzelnen Grundfarben an (0 = dunkel, 1 = hell).

Nun fehlt nur noch der Farbbefehl für die Grafikausgabe:

COLOR Reg

Hier wird also nur noch das gewünschte Farbregister angegeben, dessen Inhalt zuvor festgelegt wurde. Alle folgenden Grafikbefehle verwenden dann diese Farbe. Allerdings gibt es noch drei weitere Definitionsbefehle für Füllmuster, Grafiktext und Markierungspunkte, die ebenfalls eine Farbnummer erwarten (DEFFILL, DEFMARK und DEFTEXT).

Für den Bildschirmhintergrund und den Bildschirmrand ist stets das Farbregister 0 zuständig, während das Register 1 die Farbe enthält, mit der zum Beispiel der mit PRINT auszugebende Text dargestellt wird. Ebenso bestimmt dieses Register, in welcher Farbe Alarm-Boxen oder die Fileselect-Box erscheinen.

6.2.1 GRAPHMODE bei der Farbgrafik

Der Befehl »GRAPHMODE« bestimmt für nahezu alle Grafikbefehle den Darstellungsmodus. Der Darstellungsmodus bestimmt das Verhalten der auf dem Bildschirm vorhandenen Pixel, wenn neue Pixel darübergezeichnet werden sollen. Nun haben wir schon von den verschiedenen Bildschirmauflösungen berichtet. Es stehen also je nach Auflösung bis zu vier Bildebenen (Planes) zur Verfügung. Jeder Bildpunkt kann somit durch bis zu 4 Bit Bildschirmspeicher definiert sein, die die Farbe des Punktes bestimmen. Im Modus 640 x 400 Punkte wird jeder Punkt durch je 1 Bit dargestellt, im Modus 640 x 200 Punkte durch je 2 Bit und im Modus 320 x 200 Punkte durch jeweils 4 Bit.

Nun gerät man leicht durcheinander, wenn der Befehl »GRAPHMODE« zum Beispiel den XOR-Modus einschaltet und neue Grafikausgaben auf dem Bildschirm erfolgen sollen. Es werden jetzt nämlich alle vorhandenen Punkte mit den neu zu zeichnenden Exclusiv-ODER verknüpft. Die Wahrheitstabelle kennen Sie sicher, doch wie wird diese an den verschiedenen Bildschirm-Planes angewandt? Nun, am besten versteifen Sie Sich hierbei nicht auf die möglichen verschiedenen Planes, sondern betrachten einfach die zu verknüpfenden Farben. Dann ist die ganze Sache nämlich sehr einfach. Wir zeigen Ihnen jetzt die verschiedenen Modi anhand von Beispielen.

Es stehen vier Möglichkeiten zur Verfügung:

GRAPHMODE 0 = Replace-Modus
GRAPHMODE 1 = Transparent-Modus
GRAPHMODE 2 = XOR-Modus
GRAPHMODE 3 = Invers-Transparent-Modus

Der Modus 0 ist standardmäßig nach Programmstart eingestellt. Hier werden bei den Grafikfunktionen alle Punkte durch die darübergeschriebenen ersetzt.

Beispiele für den Modus 0:

Alte Farbe	Zeichenfarbe	Neue Farbe
1	0	0
0	1	1
4	9	9

Der Modus 1 ist der sogenannte Transparent-Modus. Hierbei werden die Farben der alten und der zu zeichnenden Punkte bitweise ODER-verknüpft. Dabei kann es auch passieren, daß die neue Farbe eine ganz andere ist als die alte oder die zu zeichnende Farbe.

Beispiele für ODER (Modus 1):

Alte Farbe	Zeichenfarbe	Neue Farbe
1	0	1
0	1	1
4	9	13

Der Modus 2 heißt XOR-Modus. Hierbei werden die Farben der alten und der zu zeichnenden Punkte bitweise Exclusiv-ODER-verknüpft.

Beispiele für XOR (Modus 2):

Alte Farbe	Zeichenfarbe	Neue Farbe
1	0	1
0	1	1
4	9	13
7	10	13
11	3	8

Die Exclusiv-ODER-Verknüpfung hat einen besonderen Effekt: Wird eine Zahl zweimal hintereinander mit derselben neuen Zahl Exclusiv-ODER-verknüpft, so ist die Ursprungszahl nach der zweiten Verknüpfung unverändert. Auch dazu ein Beispiel:

Alte Farbe	Zeichenfarbe	1.XOR	2.XOR	3.XOR
7	10	13	7	13

Man nutzt die Exclusiv-ODER-Verknüpfung auch gerne dazu, Daten zu verschlüsseln. Es werden dabei einfach die zu verschlüsselnden Daten mit einem Codewort XOR-verknüpft. Zum Entschlüsseln nutzt man dasselbe Codewort und macht wieder eine XOR-Verknüpfung. Als Ergebnis erscheinen die alten Daten! Auf unseren Bildschirm bezogen, hat das ebenfalls einen angenehmen Effekt. Es können so nämlich neue Zeichnungen gemacht werden, die, falls sie nicht stehenbleiben sollen, spurlos wieder verschwinden, wenn sie nochmals im XOR-Modus gezeichnet werden. Diese Vorgehensweise wird in Grafikprogrammen häufig angewandt, wenn zum Beispiel neue Objekte mit der Maus definiert werden sollen.

6.2.2 Beispielprogramme

Mit dem folgenden Programmbeispiel möchten wir Ihnen einmal die Farbmöglichkeiten des Atari ST zeigen. Sie finden das Programm unter dem Namen 6_1.GFA auf der beiliegenden Diskette. Dieses Programm ist selbstverständlich nur dann sinnvoll, wenn ein Farbmonitor angeschlossen ist. Aus diesem Grund wird zu Programmbeginn auch zunächst die Bildschirmauflösung ermittelt, die ja bekanntlich bei angeschlossenem Scharzweiß-Monitor immer den Wert 2 hat (hohe Bildschirmauflösung). In diesem Fall macht das Programm den Anwender darauf aufmerksam und bricht dann ab. Wir haben sechs kurze Beispielroutinen entwickelt, die alle möglichen Farben des Atari ST auf dem Bildschirm darstellen (natürlich nur jeweils 16 verschiedene Farben gleichzeitig). Die Demoprogramme laufen größtenteils selbständig ab. Lediglich beim letzten Beispiel müssen Sie mit der Maus und der Tastatur einige Angaben machen. Das komplette Listing wäre an dieser Stelle zu umfangreich. Dafür drucken wir aber das Grundprogramm ab (mit den »eingeklappten« Prozeduren). Lediglich die letzte Prozedur »farbregister_reset« ist komplett abgedruckt.

```
' *** PROGRAMM 6.1 *** ---------         EINIGE BEISPIELE FÜR FARBGRAFIK
'
aufloesung%=XBIOS(4)                    !Bildschirmauflösung ermitteln
IF aufloesung%>1
  ALERT 3,"Dieses Programm läuft nur|mit angeschlossenem|
          Farb-Monitor.",1,"ENDE",var
  END
ENDIF
IF aufloesung%=1                        !Bei mittlerer Auflösung:
  ALERT 3,"Mittlere Auflösung!|Bitte im Desktop auf|niedrige
           Auflösung|umschalten.",1,"ENDE",var
  END
```

```
ENDIF
'
farbdemo1                                 !1. Demo
farbdemo2                                 !2. Demo
farbdemo3                                 !3. Demo
farbdemo4                                 !4. Demo
farbdemo5(0.5)                            !5. Demo
farbdemo6                                 !6. Demo
' ------------------------------------- FARBEN WIEDERHERSTELLEN
farbregister_reset
END
'
> PROCEDURE farbdemo1                     !Kreise
> PROCEDURE farbdemo2                     !Durchlaufende Farben
> PROCEDURE farbdemo3                     !Durchlaufende Farben
> PROCEDURE farbdemo4                     !Tropfender Wasserhahn
> PROCEDURE farbdemo5(verz)               !Farbige Bildumrandung
> PROCEDURE farbdemo6                     !Dreidimensionales Farbdemo
PROCEDURE farbregister_reset
  ' Farbregister mit den Standardwerten belegen
  DATA 7,7,7,0,0,0,7,0,0,0,7,0,0,0,7,0,0,5,5,2,0,0,5,0
  DATA 5,5,5,2,2,2,0,7,7,0,5,5,7,0,7,5,0,5,7,7,0,5,5,0
  FOR reg%=0 TO 15
    READ rot,gruen,blau
    VSETCOLOR reg%,rot,gruen,blau
  NEXT reg%
RETURN
```

Zu Beginn des Programms wird zunächst die aktuelle Bildschirmauflösung ermittelt. Wir machen das mittels einer XBIOS-Funktion »(XBIOS(4))«. Diese liefert einen Wert zurück, der die Bildschirmauflösung angibt:

Wert	Auflösung
0	niedrige Auflösung
1	mittlere Auflösung
2	hohe Auflösung

Sollte die Auflösung größer als 1 sein, ist nur ein Schwarzweiß-Monitor angeschlossen. Dann wäre das Programm natürlich sinnlos. Deshalb wird es in diesem Fall auch sofort abgebrochen. Ebenso erscheint ein Hinweis, wenn nur die mittlere Auflösung eingestellt ist. Nun könnte man natürlich aus dem Programm heraus entsprechend umschalten. Dabei

ändert sich auch die Auflösung. Allerdings haben Sie dann auch weiterhin nur noch vier Farben zur Verfügung.

Im Programm folgen dann die sechs Farbdemos. Eine genaue Erläuterung des Programmtextes erübrigt sich hier wohl. Im übrigen sind die Routinen im Programmtext erläutert. Wer also sehen möchte, wie die Farben dargestellt werden, sollte das Programm einladen und auflisten.

Die ersten fünf Beispiele laufen nach demselben Prinzip ab: Es werden Figuren auf dem Bildschirm gezeichnet. Dabei erhält jede Figur eine andere Farbe. Anschließend werden die Inhalte der Farbregister geändert. Dadurch kann es zu einer regelrechten »Bewegung« auf dem Bildschirm kommen. So wird zum Beispiel der »Tropfen« im vierten Demo nicht ständig neu gezeichnet, sondern lediglich einmal. Allerdings in 14 verschiedenen Positionen und in ebenso vielen Farben. Davon sieht man jedoch nichts, da alle Farbregister mit selben Wert geladen sind (hier die Farbe für Blau). Anschließend werden die einzelnen Farbregister auf Weiß gesetzt und wieder auf Blau. Dadurch ergibt sich der Eindruck eines fallenden Punktes. Damit man auch merkt, wann der Punkt unten angekommen ist, liefern ein SOUND- und ein WAVE-Befehl noch einen entsprechenden Ton. Sie sehen, durch einfaches Ändern der Farbregisterinhalte kann schon Bewegung auf den Bildschirm gebracht werden. Von dieser Möglichkeit machen übrigens auch einige andere Farbdemos Gebrauch.

Ähnlich arbeiten die zwei Demos 5.1 und 5.2. Dort erhält der Bildschirm einen Rahmen aus verschiedenen Farben. Diese »laufen« nun durch Ändern der Farbregisterinhalte scheinbar durch. Dieses Prinzip eignet sich hervorragend für eigene Programme. Sie können so zum Beispiel Ihren Spielprogrammen den rechten »Rahmen« verleihen.

Nach den Beispielprogrammen folgt noch eine Routine, die die Farbregister wieder mit Standardwerten belegt. Natürlich können Sie diese »Standardwerte« auch Ihren ganz speziellen Wünschen anpassen. Die Routine könnte dann vor jedem Farbprogramm sinnvoll eingesetzt werden.

Zum letzten Farbdemo noch eine Anmerkung. Wir haben dort eine dreidimensionale Grafik mit Hilfe von zwei Farben dargestellt. Um den *3-D-Effekt* zu nutzen, müssen Sie eine sogenannte Rot-Grün-Brille aufsetzen. Es handelt sich bei diesem Verfahren um eine gängige Methode der dreidimensionalen Darstellung auf einem zweidimensionalen Bild. Wir gehen daher davon aus, daß bei Ihnen oder in Ihrem Bekanntenkreis eine derartige Brille zur Verfügung steht.

Das Prinzip der 3-D-Darstellung mit zwei Farben besteht darin, daß zwei Kameras aus verschiedenen Blickwinkeln dasselbe Motiv aufnehmen. Die zwei Bilder werden dann gleichzeitig auf einem Monitor angezeigt. Eines der Bilder zeigt das Motiv in roter Farbe, das andere in grüner Farbe (in unserem Beispiel ist Rot links und Grün rechts). Wird nun die rote Farbe nur vom linken Auge gesehen und die grüne Farbe nur vom rechten Auge, verarbeitet das Gehirn die getrennten Informationen zu einem räumlichen Bild. Das Problem besteht natürlich in der einwandfreien Trennung der zwei Bilder, die ja gleichzeitig dar-

gestellt werden müssen. Nach Möglichkeit sollte jedes Auge auch wirklich nur das für dieses Auge bestimmte Bild sehen. Zu diesem Zweck dient die besagte Brille. Diese hat nämlich zwei verschiedenfarbige Filter. Das eine Glas läßt nur die rote Farbe passieren und das andere nur die grüne Farbe. Somit ist eine relativ gute Trennung der beiden Bilder möglich.

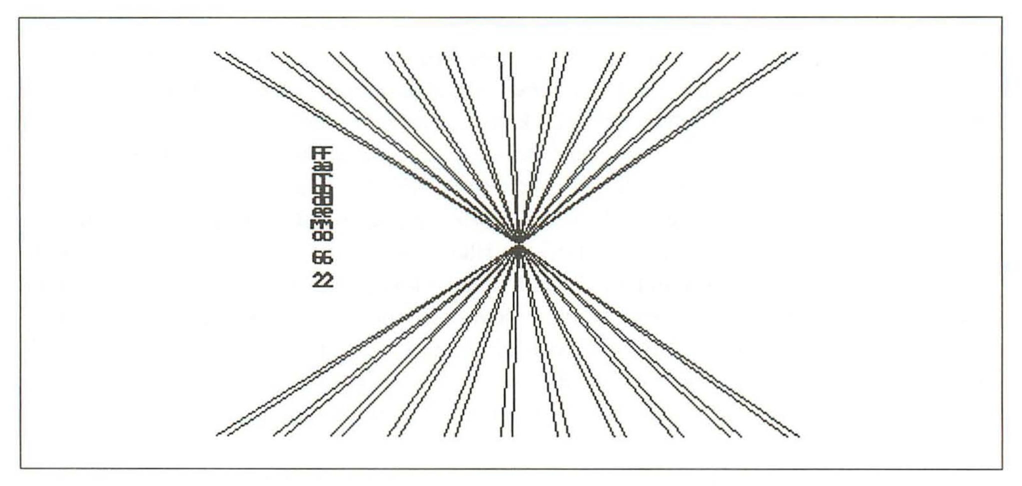


Bild 6.5: Bild des Farbdemos »6_1.GFA«

Der sich einstellende 3-D-Effekt ist ziemlich verblüffend. Allerdings kann es möglich sein, daß man die Bilder eine Weile ansehen muß, bis man das Räumliche wahrnimmt. Auch kann der Effekt noch durch Farb- oder Helligkeits-Einstellung des Monitors verbessert werden. (Oft gibt es einen guten Effekt, wenn der Monitor sehr hell eingestellt wird.)

In diesem Buch stellen wir Ihnen noch ein zweites Programm vor, das ebenfalls Rot-Grün-Bilder erstellen kann. Es handelt sich dabei um ein Programm zur Berechnung der sogenannten Apfelmännchen-Bilder (Fraktale). Dort ist der Effekt leider nicht so schön wie bei unseren Farbdemos, weil die Bilder berechnet und nicht speziell für die Rot-Grün-Darstellung entwickelt werden.

Es gibt übrigens noch weitere Methoden der dreidimensionalen Darstellungen mittels Computer. Dabei spielt meist eine Brille eine entscheidende Rolle. Neben der Rot-Grün-Brille gibt es noch Brillen, die Filter für polarisiertes Licht enthalten. Werden nun die Brillengläser und das polarisierte Licht der darzustellenden Bilder um 90 Grad gedreht, können die Bilder einwandfrei getrennt werden. Eine andere Möglichkeit nutzt Brillengläser mit einer LCD-Schicht. Sie kennen ja die LCD-Anzeigen aus Taschenrechnern und Uhren: Wird eine Spannung angelegt, wird die Anzeige dunkel. Ebenso werden die Brillengläser lichtundurchlässig, wenn die Spannung anliegt. Werden nun abwechselnd das linke und das rechte Glas durchlässig, während auf dem Computerbildschirm abwechselnd das Bild für links und rechts angezeigt wird, ergibt sich der 3-D-Effekt. Bei dieser Methode

flimmert das Bild leider relativ stark, da die Umschaltfrequenz zu niedrig ist. Dafür können hierbei auch farbige Bilder dargestellt werden.

6.3 Die Grafikbefehle

Daß ein Computer mit den außergewöhnlichen Grafikfähigkeiten, wie sie der Atari ST besitzt, auch von den Programmiersprachen her durch entsprechende Befehle unterstützt werden muß, ist klar. Was GFA-Basic auf diesem Gebiet anbietet, ist jedoch noch einiges mehr, als man von anderen Programmiersprachen gewohnt ist. Da stehen fast 50 »normale« Grafikbefehle zur Verfügung. Außerdem können natürlich alle VDI-Routinen genutzt werden. Zusätzlich zu diesen Befehlen gibt es auch noch diverse sogenannte Line-A-Aufrufe. Dazu gleich einige Worte: Bei den Line-A-Routinen handelt es sich um eine besondere Art der Maschinensprache-Befehle des MC 68000, also des Prozessors im Atari ST. Alle Maschinensprache-Befehle, die mit »$A« beginnen, bewirken einen Interrupt (hier Exception) des Mikroprozessors. Es wird dann eine der sogenannten Exceptionroutinen ausgeführt. Das können beliebige Programme sein. Hier sind es jedoch Grafikroutinen. Diese werden auch vom Betriebssystem des Rechners ausgiebig genutzt. Man kann sie somit auch als Grafikkern bezeichnen. Der Vorteil der Routinen liegt darin, daß sie sehr schnell sind. Werden nun statt der VDI-Aufrufe die Line-A-Routinen genutzt, kann das erhebliche Geschwindigkeitsvorteile bringen. Ihnen bringen sie auf jeden Fall die Qual der Wahl, welche Grafikbefehle genutzt werden sollen, und außerdem die Schwierigkeiten, die sich aus den Nichtkompatibilitäten der gleichzeitigen Verwendung von Line-A und VDI ergeben.

Es folgen nun in Kurzform die Grafikbefehle des GFA-Basic.

BITBLT source%(),destination%(),parameter%()

Rechteckige Bildausschnitte kopieren.

BOX x1,y1,x2,y2

Stellt ein Rechteck mit den angegebenen Koordinaten dar.

BOUNDERY wert

Schaltet die Umrandung ein (wert<>0) oder aus (wert=0).

CIRCLE x,y,r[,w1,w2]

Stellt einen Kreis dar.

CLIP x,y,breite,hoehe[OFFSET x0,y0]
CLIP x1,y1 TO x2,y2[OFFSET x0,y0]
CLIP #n[OFFSET x0,y0]
CLIP OFFSET x0,y0
CLIP OFF

Ausgabebereich für die Grafikbefehle festlegen.

CLS

Löscht den Bildschirm.

DEFFILL [farbe],[stil],[muster]
DEFFILL [farbe],muster$

Füllmuster bestimmen.

DEFLINE [stil],[dicke],[anfang,ende]

Linienart bestimmen.

DEFMARK [farbe],[art],[groesse]

Markierungspunkte einstellen.

DEFMOUSE maus
DEFMOUSE muster$

Mauscursor einstellen.

DEFTEXT [farbe],[attribut],[winkel],[hoehe],[fontnummer]

Textstil bestimmen.

DRAW ausdruck

Zeichnet eine Grafik.

wert=DRAW(nummer)

Liefert Informationen über den imaginären Zeichenstift.

DRAW x1,y1[TO x2,y2][TO x3,y3]...
DRAW [TO][x,y]

Zeichnet einen Punkt oder eine oder mehrere Linien.

ELLIPSE x,y,rx,ry[w1,w2]

Stellt eine Ellipse oder ein Ellipsensegment dar.

FILL x,y[,farbe]

Füllt eine abgeschlossene Fläche.

GRAPHMODE mode

Bestimmt den Grafikmodus für die meisten Grafikbefehle.

LINE x1,y1,x2,y2

Zeichnet eine Linie.

PBOX x1,y1,x2,y2

Zeichnet eine gefüllte Box.

PCIRCLE x,y,r[,w1,w2]

Zeichnet einen ausgefüllten Kreis oder ein Kreissegment.

PELLIPSE x,y,rx,ry[,w1,w2]

Zeichnet eine ausgefüllte Ellipse oder ein Ellipsensegment.

PLOT x,y

Zeichnet einen Punkt. Entspricht DRAW x,y.

wert=POINT(x,y)

Ermittelt die Farbe eines Punktes.

POLYLINE anzahl,x(),y()[OFFSET xoffs,yoffs]

Stellt ein Vieleck mit »anzahl« Ecken.

POLYFILL anzahl,x(),y()[OFFSET xoffs,yoffs]

Ein gefülltes Vieleck darstellen.

POLYMARK anzahl,x(),y()[OFFSET xoffs,yoffs]

Markiert ein Vieleck mit »anzahl« Ecken.

PUT xpos,ypos,bild$

Zeichnet einen rechtwinkligen Bildteil.

RBOX x1,y1,x2,y2

Zeichnet ein Rechteck mit abgerundeten Ecken.

SETCOLOR reg,rot,gruen,blau
SETCOLOR reg,mischwert

Definiert die Farbe eines Farbregisters.

SETDRAW xpos,ypos,winkel

Kurzform des Befehls »DRAW "MA",x,y,"TT",w«.

SGET bild$

Speichert den gesamten Bildschirm in eine Stringvariable.

SPRITE muster$[,x,y]

Stellt ein Sprite (Spritedaten in »muster$«) dar.

SPUT bild$

Kopiert den Inhalt des Strings »bild$« auf den Bildschirm.

TEXT x,y,[,laenge],text

Stellt einen Grafiktext an der angegebenen Position dar.

VSYNC

Auf einen senkrechten Strahlrücklauf des Monitors warten.

Line-A-Routinen

ACHAR ascii,x,y,groesse,stil,winkel

Ausgabe eines Zeichens mit dem angegebenen ASCII-Code.

ACLIP flag,xmin,ymin,xmax,ymax

Setzt das Ausgaberechteck für die Line-A-Grafikausgaben.

ALINE x1,y1,x2,y2,farbe,muster,modus

Zeichnet eine Linie mit dem »muster«.

APLOY adresse,anzahl,y0 TO y1,farbe,modus,m_adresse,muster_anz

Zeichnet ein Vieleck ohne Begrenzungslinie und füllt dieses.

ARECT x1,y1,x2,y2,farbe,m_adresse,muster_anz

Stellt ein Rechteck dar.

ATEXT x,y,font,text$

Gibt einen Text aus.

BITBLT adresse%
BITBLT x%()

BITBLT-Routine der Line-A-Routinen. Die Bedeutung der Parameter entnehmen Sie bitte Ihrer GFA-Basic-Anleitung.

HLINE x1,y,x2,farbe,modus,m_adresse,muster_anz

Zeichnet waagerechte Linien.

L~A

Ermittelt die Basisadresse der Line-A-Variablen.

PSET x,y,farbe

Setzt einen Punkt auf dem Bildschirm.

farbe=PTST(x,y)

Ermittelt die Farbe des Punktes.

6.3.1 Die Befehle GET und PUT

Zwei Grafikbefehle des GFA-Basic verdienen es, in einem Abschnitt besonders beschrieben zu werden. Es handelt sich um die Befehle, die es ermöglichen, rechteckige Bildausschnitte in Stringvariablen zu speichern und bei Bedarf wieder auf den Bildschirm zu bringen. Die Befehle »GET« und »PUT« ermöglichen das.

Bei einem grafikorientierten System wie dem Atari ST, der zudem noch eine recht hohe Bildschirmauflösung bietet, werden häufig Routinen benötigt, die Bildschirm- oder Speicherbereiche in andere Bildschirm- oder Speicherbereiche kopieren. Sie haben das schon in Form vom *Bit-Block-Transfer* (BITBLT) kennengelernt. Ihnen stehen im GFA-Basic gleich drei Versionen des Befehls »BITBLT« zur Verfügung. Auch »RC_COPY« kann zum Kopieren von Bildschirmausschnitten genutzt werden (siehe auch Kapitel 7). Neben dem eigentlichen Kopieren bieten diese Routinen auch noch verschiedene Verknüpfungen zwischen Quell- und Zielbereich. Allerdings müssen, bedingt durch die vielseitigen Möglichkeiten des »BITBLT«, recht umfangreiche Parameterblöcke übergeben werden (siehe dazu Kapitel 4 und unser Programmlisting 7_1.PRG).

GFA-Basic bietet für das Speichern und Wiederdarstellen von Bildschirmausschnitten zwei besondere Befehle, die mit nur wenigen Parametern auskommen. Darüber hinaus braucht sich der Anwender keine Gedanken über die Speicherreservierung zu machen (sofern noch genügend Speicherraum für Variablen zur Verfügung steht). Die Bildschirmdaten werden nämlich in einfache Stringvariablen übernommen.

Dies geschieht durch:

GET x1,y1,x2,y2,string$

Sie geben also die gegenüberliegenden Bildschirmkoordinaten des Rechtecks sowie eine Stringvariable an. GFA 3.0 wird dann in dieser Variablen ein Bitmuster ablegen. Später kann dieses mit

PUT x,y,string$,modus

wieder an beliebiger Stelle auf dem Bildschirm dargestellt werden. Natürlich können Sie so viele Bildausschnitte speichern, wie Sie möchten bzw. Speicherplatz zur Verfügung haben.

Der Parameter »modus« bestimmt, in welchem Verknüpfungsmodus das gespeicherte Bitmuster mit dem vorhandenen Bildschirminhalt verknüpft werden soll. Ihnen stehen dazu 16 Möglichkeiten offen (0 bis 15). In der folgenden Tabelle listen wir die verschiedenen Modi auf.

Modus	Verknüpfung
0	Zielbereich löschen
1	Quellbereich AND Zielbereich
2	Quellbereich AND (NOT Zielbereich)
3	Zielbereich = Quellbereich
4	(NOT Quellbereich) AND Zielbereich
5	Zielbereich unverändert
6	Quellbereich XOR Zielbereich
7	Quellbereich OR Zielbereich
8	NOT (Quellbereich OR Zielbereich)
9	NOT (Quellbereich XOR Zielbereich)
10	Zielbereich invertieren
11	Quellbereich OR (NOT Zielbereich)
12	Zielbereich = (NOT Quellbereich)
13	(NOT Quellbereich) OR Zielbereich
14	NOT (Quellbereich AND Zielbereich)
15	Alle Punkte des Zielbereichs setzen

Damit Sie diese Möglichkeiten und ihre Auswirkungen auf den Bildschirminhalt auch einmal auf Ihrem Bildschirm ausprobieren können, haben wir ein kleines Beispielprogramm geschrieben, das alle Darstellungsmöglichkeiten verdeutlicht. Es handelt sich um das Programm 6_5.GFA, das Sie natürlich auch auf der beiliegenden Diskette finden. Wir drucken es hier außerdem ab, da es nicht sehr umfangreich ist.

```
' *****                    PROGRAMM 6.5                     *****
' ****************************************************************
' *****                                                     *****
' *****          BEISPIELPROGRAMM ZUR DARSTELLUNG           *****
' *****        DER VERKNÜPFUNGSMODI BEI GET UND PUT         *****
' *****                                                     *****
' *****           WILHELM BESENTHAL UND JENS MUUS           *****
' *****                                                     *****
' *****            MARKT & TECHNIK, MÜNCHEN 1988            *****
' *****                                                     *****
' ****************************************************************
'
'
'
variablen
bildschirm_zeichnen
REPEAT
  REPEAT
```

```
    t%=MOUSEK
  UNTIL t%>0
  ADD modus%,m%(t%)
  modus%=MAX(MIN(modus%,15),0)
  ausgabe
UNTIL t%=3
END
'
```

Das Programm besteht also lediglich aus einem kurzen Hauptprogramm und drei Prozeduren, die einige Variablen deklarieren, den Grundbildschirm (mit einigen Füllmustern) zeichnen und die eigentliche Verknüpfung darstellen. Der Grundbildschirm wird lediglich einmal gezeichnet und dann mit dem Befehl »SGET bild$« gespeichert, einer besonderen Variante von »GET«, die nur ganze Bildschirme in einer Stringvariablen speichern kann (dafür allerdings sehr schnell). In der Prozedur »ausgabe« wird dann dieser Bildschirminhalt mit »SPUT bild$« wieder dargestellt. Anschließend erfolgt die Verknüpfung des Bildschirminhalts mit dem Bitmuster in »ausschnitt$«.

```
PROCEDURE variablen                          !Wichtige Variablen deklarieren
  DIM m%(3),modus$(15)
  FOR i%=1 TO 3
    READ m%(i%)                              !Wird je nach Maustaste zu Modus addiert
  NEXT i%
  FOR i%=0 TO 15
    READ modus$(i%)                          !Modus im Klartext für Bildschirmausgabe
  NEXT i%
  DATA 1,-1,0
  DATA "Ziel löschen","Quelle AND Ziel","Quelle AND (NOT Ziel)"
  DATA "Ziel = Quelle","(NOT Quelle) AND Ziel","Ziel unverändert"
  DATA "Quelle XOR Ziel","Quelle OR Ziel","NOT (Quelle OR Ziel)"
  DATA "NOT (Quelle XOR Ziel)","NOT Ziel","Quelle OR (NOT Ziel)"
  DATA "Ziel = (NOT Quelle)","(NOT Quelle) OR Ziel","NOT (Quelle AND Ziel)"
  DATA "Ziel = 1"
RETURN
PROCEDURE bildschirm_zeichnen                !Bildschirm vorbereiten
  CLS
  DEFFILL 1,1
  PBOX 0,0,319,199
  DEFFILL 1,2,7
  PBOX 320,200,639,399
  PRINT AT(41,1);"Linke Maustaste  = Modus+1"
  PRINT AT(41,2);"Rechte Maustaste = Modus-1"
  PRINT AT(1,15);"Originalausschnitt"
  DEFLINE 1,1,0,0
```

```
  BOX 0,250,150,399
  FOR i%=0 TO 100 STEP 50
    DRAW i%,250 TO i%+25,399 TO i%+50,250
    DEFFILL 1,i%\50,1
    FILL i%+10,390
    DEFFILL 1,i%\50+1,4
    FILL i%+10,260
  NEXT i%
  GET 0,250,150,399,ausschnitt$
  SGET ganzes_bild$
RETURN
PROCEDURE ausgabe                         !Verknüpfungsmodus darstellen
  SPUT ganzes_bild$
  PRINT AT(41,3);"Modus                   = ";modus%
  PRINT AT(41,4);"Verknüpfung             = ";modus$(modus%);
  FOR i%=0 TO 500 STEP 50
    PUT i%,i%/2,ausschnitt$,modus%
  NEXT i%
RETURN
```

Steuern können Sie das Programm mit den Maustasten. Dabei bewirkt ein Druck der linken Taste, daß der Modus um 1 erniedrigt wird, und ein Druck der rechten Taste, daß der Modus um 1 erhöht wird.

Die Programmzeile

modus%=MAX(MIN(modus%,15),0)

verhindert, daß Werte kleiner als 0 und größer als 15 erscheinen.

Wenn Sie beide Maustasten gleichzeitig drücken, wird das Beispielprogramm beendet. Der mit GET gebildete String hat folgenden Aufbau:

1. Byte:	Anzahl der Punkte in X-Richtung (Highbyte)
2. Byte:	Anzahl der Punkte in X-Richtung (Lowbyte)
3. Byte:	Anzahl der Punkte in Y-Richtung (Highbyte)
4. Byte:	Anzahl der Punkte in Y-Richtung (Lowbyte)
5. Byte:	Bildschirmauflösung (Highbyte)
6. Byte:	Bildschirmauflösung (Lowbyte)
7. bis n. Byte:	Bitmuster des Bildschirmausschnitts

Die Bildschirmauflösung hat den Wert 1 für die hohe Auflösung, 2 für die mittlere und 3 für die niedrige Auflösung.

Der Bildschirminhalt ist zeilenweise und wortweise, also jeweils 2 Byte (16 Bit), gespeichert. Wenn also zum Beispiel mit »GET« ein Ausschnitt in der hohen Auflösung mit einer Länge von 17 Punkten und einer Höhe von 10 Punkten gespeichert werden soll, ist der

gebildete String 46 Byte lang. Die ersten 6 Byte geben dabei die Länge und Höhe sowie die Bildschirmauflösung an. Anschließend folgen die Bitmuster für die 10 Zeilen mit jeweils 17 Punkten. Da diese Speicherung ja wortweise erfolgt, werden für jede Zeile zwei Wörter, also 4 Byte, benötigt. Im ersten Wort stehen die ersten 16 Punkte und im zweiten Wort nur ein Punkt. Die restlichen 15 Bit werden ignoriert.

6.4 Räumliche Darstellungen

Im Abschnitt 6.2.2 haben Sie ein eindrucksvolles Beispiel für räumliche Darstellungen gesehen. Wir haben dort im Beispielprogramm für Farbgrafik ein sogenanntes Rot-Grün-Bild erstellt, das, mit einer besonderen Brille betrachtet, tatsächlich einen verblüffenden räumlichen Effekt bringt. Im weiteren Verlauf dieses Kapitels werden wir uns mit Schwarzweiß-Grafiken beschäftigen. Natürlich sind dabei keine räumlichen Darstellungen wie bei den Rot-Grün-Bildern möglich. Man bedient sich daher anderer Methoden, die nur eine scheinbare räumliche Darstellung ermöglichen. Es sind die Darstellung mit Fluchtpunkt und die Kavaliersperspektive. Die folgenden beiden Bilder zeigen diese zwei Darstellungsformen. Rechts sehen Sie jeweils ein Beispiel dazu. Übrigens folgen in diesem Kapitel noch die entsprechenden Programme, die diese Bilder berechnet haben.

In Bild 6.6 (Darstellung mit Fluchtpunkt) sehen Sie, daß die entfernteren Teile eines Körpers kleiner wirken als die näher zum Betrachter liegenden. Dabei zeigen die parallel zueinander liegenden Geraden auf einen entfernten Punkt, den Fluchtpunkt. Dagegen ist der Fluchtpunkt in Bild 6.7 unendlich weit entfernt. Die Kanten des Würfels werden somit auch parallel dargestellt. Damit man mehr als nur die Vorderseite des Würfels sieht, werden die weiter entfernten Teile des Objekts zur Seite versetzt gezeichnet. In der Regel wird dabei zur einfacheren Darstellung ein Winkel von 45 Grad verwendet. Aber es geht auch anders, wie das Beispiel rechts zeigt.

Durch die vorstehenden Darstellungsmöglichkeiten sind keine dreidimensionalen Effekte wie bei den Rot-Grün-Bildern möglich. Stellen Sie sich vor, Sie könnten nur mit einem Auge sehen. Dann sind keine räumlichen Wahrnehmungen mehr möglich. Es fehlt eine Hälfte der Bildinformationen, die allerdings nahezu ersetzt werden kann. Nämlich dann, wenn das dargestellte Objekt in schneller Folge aus verschiedenen Blickwinkeln gezeigt wird. Das menschliche Gehirn kann dann nämlich aus den »seriellen« (nacheinander ablaufenden) Informationen eine räumliche Darstellung »errechnen«.

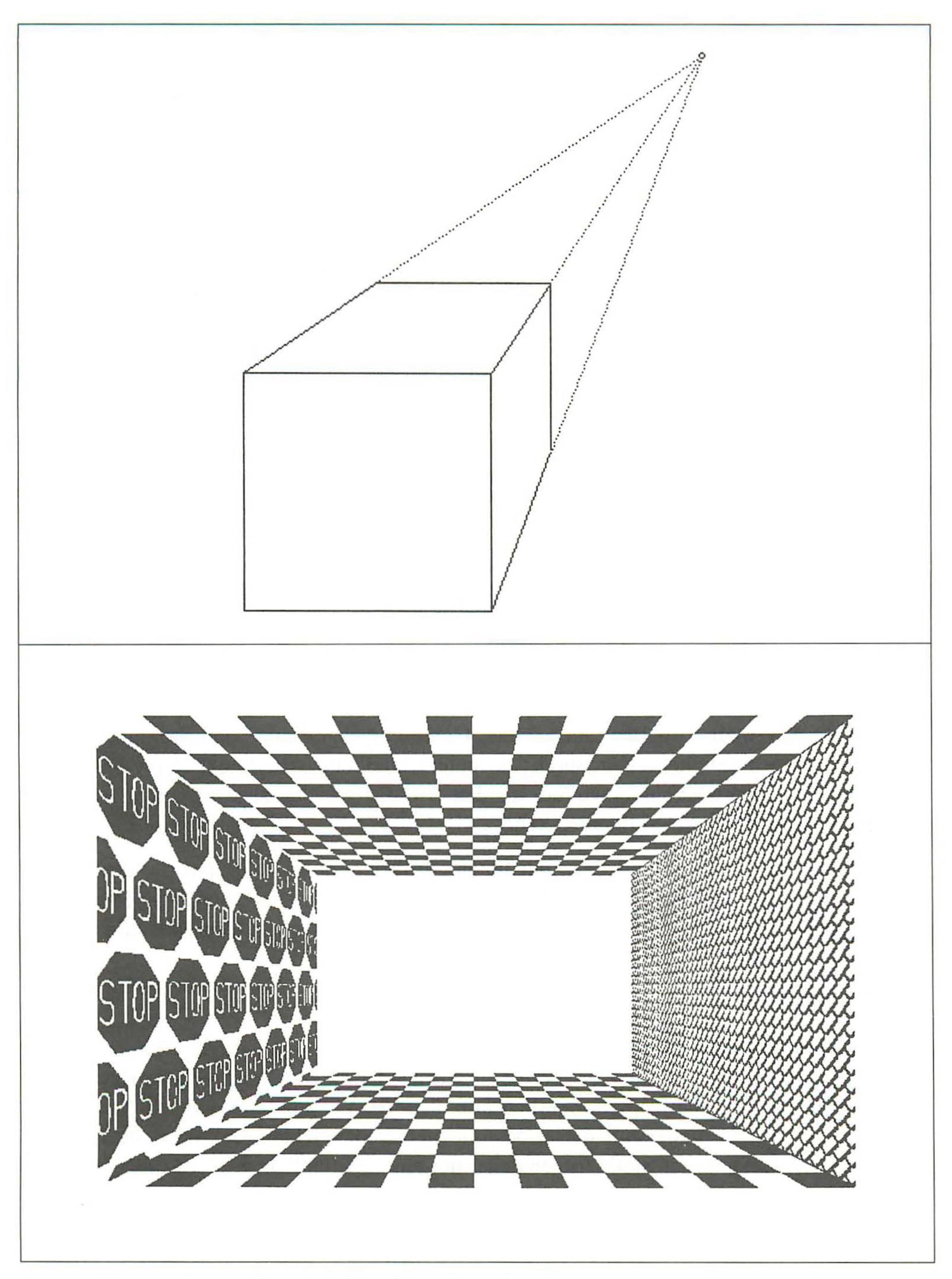


Bild 6.6: Darstellung mit Fluchtpunkt und ein Beispiel dazu (Programm 6_2)

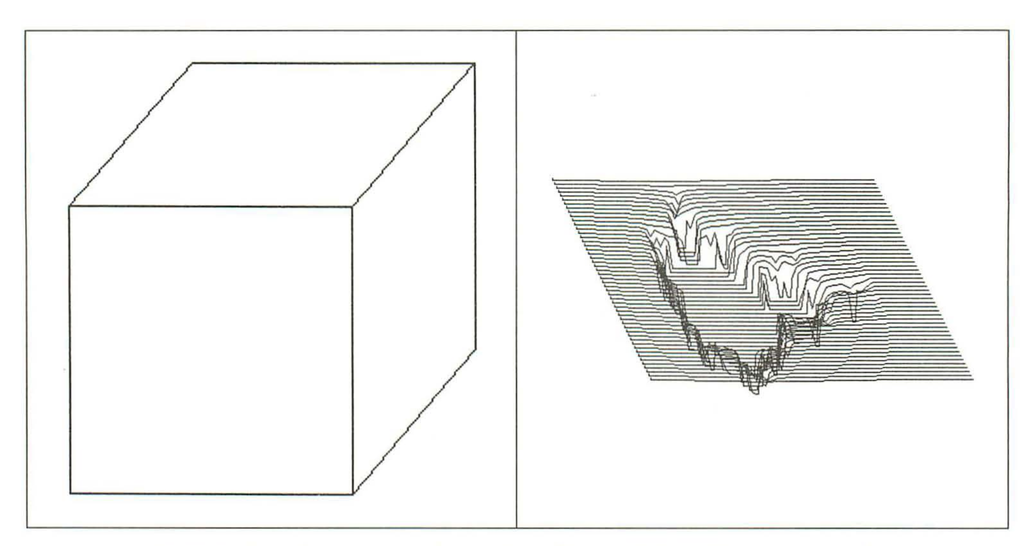


Bild 6.7: Kavaliersperspektive und ein Beispiel dazu (Programm 6_3)

So ist es auch möglich, daß Menschen, die auf einem Auge erblindet sind, trotzdem Entfernungen schätzen können. Man muß dabei lernen, das, was vorher von zwei Augen aufgenommen wurde, mit einem Auge zu sehen, und zwar aus verschiedenen Perspektiven. Dazu ist entweder die Bewegung des Kopfes oder des zu sehenden Objektes erforderlich. Natürlich spielen dabei auch Erfahrungswerte eine Rolle.

Nach diesem Prinzip arbeiten auch viele Computerprogramme, die dreidimensionale Darstellungen ermöglichen. Nur durch die unterschiedliche Abbildung der Objekte aus verschiedenen Blickwinkeln bewirken sie den 3-D-Effekt, der aber trotzdem nicht mit unserem Rot-Grün-Beispiel mithalten kann.

In den folgenden Abschnitten zeigen wir Ihnen ein Programmbeispiel, das die Darstellung mit Fluchtpunkt (Programm 6_2.GFA) zeigt, und ein zweites Beispiel, das eine Kavaliersperspektive nutzt (Programm 6_3.GFA, wahlweise auch in Rot-Grün-Darstellung).

6.4.1 Raytracing

Im folgenden Programmbeispiel zeigen wir Ihnen ein einfaches Raytracing-Programm. Vermutlich haben Sie schon etwas von dieser Art der Grafikerstellung gehört. Raytracing-Routinen sind naturgemäß außerordentlich kompliziert. Wir müssen daher alle diejenigen Leser bitten, die sich näher mit dieser Materie befassen möchten, auf spezielle Literatur zurückzugreifen. An dieser Stelle folgen daher nur einige Grundlagen.

Beim Raytracing (Strahlverfolgung) wird der Verlauf eines Lichtstrahls von seiner Entstehung (Lichtquelle) bis zu seinem Ziel (hier das Auge des Betrachters) simuliert. Bedauerlicherweise handelt es sich dabei um unendlich viele Strahlen, die verfolgt werden

müssen. Eine Berechnung würde demnach auch unendlich lange dauern und wäre daher für ein Computerprogramm wohl kaum geeignet. Man geht aus diesem Grund einen anderen Weg: Die Lichtstrahlen werden nicht ausgehend von der Lichtquelle, sondern genau umgekehrt, nämlich vom Auge des Betrachters ausgehend berechnet. Man kann diese Lichtstrahlen somit auch als »Sehstrahlen« bezeichnen. Es müssen jetzt zwar immer noch viele Strahlen, aber nicht mehr unendlich viele verfolgt, also berechnet werden. Nun muß man sich darüber im klaren sein, daß Lichtstrahlen sozusagen sehr klein sind und daß nach menschlichen Maßstäben das Auge ständig (außer bei Dunkelheit) unendlich viele (sichtbare) Lichtstrahlen erreichen. Auch diese Methode ist also für das Raytracing nicht geeignet.

Die Lösung dieses Problems liegt in der endlich großen Darstellungsmöglichkeit eines Computers. Was nützt es, unendlich viele Strahlen zu berechnen, wenn doch nur maximal 640 x 400 Punkte auf dem Bildschirm unseres Computers dargestellt werden können? Somit reicht es doch, lediglich diese 256.000 Punkte (640 x 400), hier »Zielpunkte« der Strahlen, zurückzuverfolgen. Das Ergebnis der Berechnung wird einfach auf dem Bildschirm dargestellt. Der Bildschirm wird dadurch also zum »Auge« des Betrachters.

Beim Raytracing verfolgt man also jeden Bildschirmpunkt »in die Tiefe« und »sieht« nach, worauf der so entstehende Strahl trifft. Das kann zum Beispiel eine Wand sein, die den imaginären (scheinbaren) Raum begrenzt. Möglicherweise ist innerhalb des Raumes ein Gegenstand. Dieser könnte eine bestimmte Farbe haben, die dann auch unser Lichtstrahl und somit der Bildpunkt einnimmt. Es ist aber auch möglich, daß der Strahl auf eine spiegelnde Oberfläche trifft und in eine andere Richtung weiterverfolgt werden muß (bitte entschuldigen Sie an dieser Stelle unsere sehr einfache und grobe Erläuterung dieses Phänomens). Grundsätzlich muß aber jedes Objekt innerhalb unserer »Bildschirmwelt« durch mathematische Beschreibungen definiert werden. Die Farbe des Bildpunktes wird also durch die Farbe des letzten nicht mehr reflektierenden Punktes, auf den der Sehstrahl trifft, bestimmt.

Die Berechnung wird somit also auch für Computer möglich. Bisher waren damit allerdings lediglich Großrechner beschäftigt. Die zunehmende Leistungsfähigkeit der Personalcomputer macht aber auch auf diesen Geräten derartige Strahlverfolgungen in annehmbarer Rechenzeit möglich.

Kommen wir nun zu unserem Beispielprogramm. Sie finden es als Programm 6_2.GFA auf der beiliegenden Diskette. Es ist nur im hochauflösenden Modus, also mit angeschlossenem Schwarz-Weiß-Monitor, lauffähig. Das Ergebnis der Strahlverfolgung ist demnach auch recht einfach: kein Licht oder Licht, also schwarz oder weiß. Das Programm kann aber problemlos auf Farbbetrieb umgestellt werden.

Durch unser Programm muß zunächst eine kleine »Welt« erschaffen werden. Um überhaupt etwas auf dem Bildschirm darstellen zu können, konstruieren wir einen imaginären rechtwinkligen »Raum«. Dieser hat also sechs Seiten, die wir vorteilhafterweise mit Bildern bestücken. Hier sind das jeweils Grafikseiten, die wir als Seitenflächen nutzen. Es entsteht also ein Raum, wie in Bild Bild 6.8 ersichtlich ist. Auf dem Bild erkennen Sie auch eine Figur (eine spiegelnde Kugel), die im Raum schwebt und die Lichtstrahlen reflektiert.

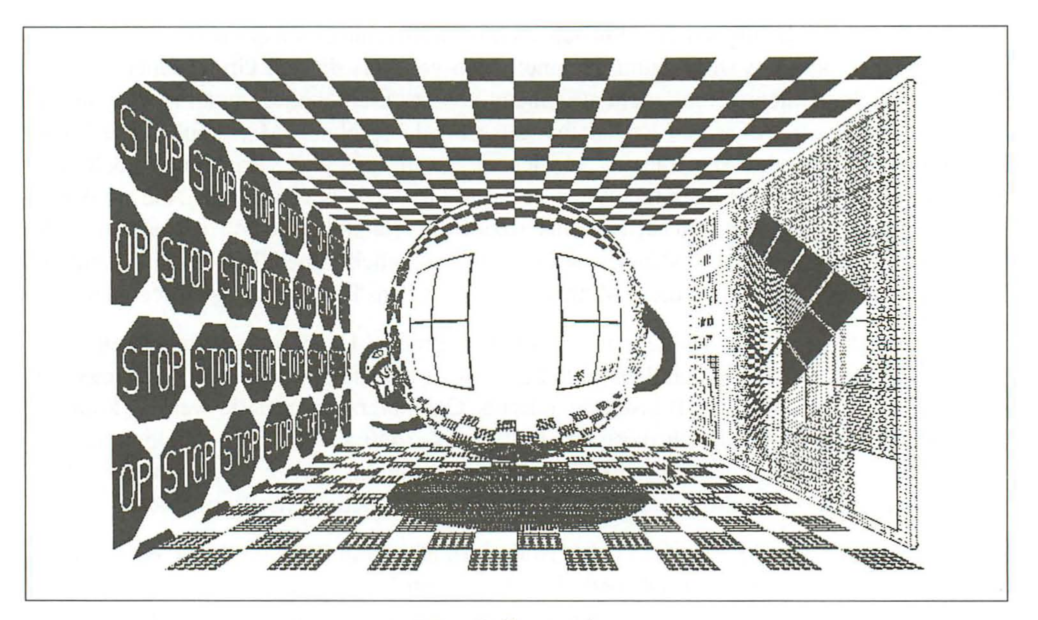


Bild 6.8: Raytracing

Der imaginäre (scheinbare) Raum besteht aus einem Rechteck mit Kantenlängen von 640 x 400 x 640 (Breite x Höhe x Tiefe) Punkten. Sie sehen daran, daß für den Boden und die Decke des Raumes kein normales Grafikbild genutzt werden kann, da dieses »nur« 640 x 400 Punkte groß ist, hier aber 640 x 640 Punkte benötigt werden. Im Programm werden daher hierfür auch keine Grafiken eingeladen, sondern spezielle größere Bilder entwickelt. Wir zeichnen in das untere 640 x 640 Punkte große Bild ein Karomuster, das Sie auch in Bild 6.8 erkennen können. Auch an der »Decke« sehen Sie das Muster. Hier handelt es sich allerdings um ein berechnetes Bild nach der Formel:

farbe=(x_koordinate/40+y_koordinate/40) AND 1

In der dargestellten Kugel spiegeln sich also die Seiten des Raumes. Natürlich kann in einem solchen künstlich konstruierten Raum auch noch mehr als eine Kugel dargestellt werden. Dann kann und wird es passieren, daß die Lichtstrahlen zwischen mehreren Kugeln hin- und herreflektiert werden, bis sie den Betrachter erreichen (bzw. umgekehrt die Lichtquelle). Der Berechnungsaufwand wird hierdurch natürlich erheblich umfangreicher und die Berechnungszeit selbstverständlich ebenfalls. Wir müssen an dieser Stelle auch darauf hinweisen, daß unser Basic-Programm eben »nur« ein Basic-Programm ist und daher bei diesen umfangreichen Berechnungen auch mitunter recht lange Ausführungszeiten benötigt. Das Kompilieren des Programms bringt eine starke Geschwindigkeitssteigerung.

Das Grundprogramm stammt übrigens nicht aus unserer eigenen Feder. Es wurde von Tom Hudson entwickelt, einem Grafikspezialisten, der die Möglichkeiten des Atari ST außerordentlich gut kennt. Er entwickelte das Programm in der Sprache C, von der wir wiederum

eine Version in Omikron-Basic vorliegen hatten. Wir haben dieses Programm auf GFA-Basic angepaßt, aber auch einige ganz spezielle GFA-Basic-Optionen eingebaut. Wir möchten uns hiermit beim Programmautoren Tom Hudson bedanken, daß wir seine Grundroutine für unser Listing nutzen durften.

Das so entstandene GFA-Basic-Programm war für uns leider recht enttäuschend. Es benötigte einfach zuviel Zeit, um ein komplettes Bild zu berechnen. Das Programmprojekt war also zumindest unkompiliert für eine sinnvolle Nutzung unbrauchbar, oder wollen Sie etwa 10 oder noch mehr Stunden auf das Ergebnis warten? Um das Programm überhaupt veröffentlichen zu können, mußte es also wesentlich schneller werden. Glücklicherweise hatten wir es hier mit einem sehr dankbaren Objekt zu tun, das auf unsere Programmoptimierungen durch enorme Geschwindigkeitsgewinne reagierte. Vielleicht ist es für den einen oder anderen interessant, zu erfahren, wie wir dieses Programm beschleunigen konnten. Deshalb an dieser Stelle eine kurze Erläuterung hierzu.

Grundsätzlich muß man sich vor jeder Programmoptimierung bezüglich der Abarbeitungsgeschwindigkeit natürlich erst einmal vergewissern, welche Programmteile besonders zeitintensiv sind. Der prozentuelle Anteil an der Gesamtausführungszeit der einzelnen Routinen oder Programmblöcke muß ermittelt werden.

Unser Raytrace-Programm besteht im wesentlichen aus folgenden Teilen:

1. Vorbereitungen (Speicher reservieren, Variablen dimensionieren usw.)
2. Einlesen der Parameter aus der Parameterdatei.
3. Laden der Bilder für die Seitenwände.
4. Darstellung der »Kugelschatten«.
5. Raytracing-Routine, bestehend aus:
   a) Routine für das gesamte Bild
   b) Routine für die Berechnung innerhalb der Kugelbereiche

Unsere ersten Versuche mit dem Programm ergaben Ausführungszeiten von 10 und mehr Stunden! Davon entfielen auf die ersten vier Punkte der Liste nur etwa 37 Sekunden. Das ist natürlich verschwindend gering. Dennoch störte z.B. die langsame Erstellung der Schatten. Durch Verwendung unserer Festplatte wurden die Punkte 2 und 3 der Liste erheblich schneller (jetzt ca. 2,5 Sekunden). Im ursprünglichem Listing war der Punkt 4 (Schatten) sehr ungünstig programmiert. Und zwar wurden die Kugeln auf dem Bildschirm gezeichnet und dann der Bildschirmspeicher und der Speicherbereich für den Boden Byte für Byte mit »PEEK« ausgelesen, ODER-verknüpft und wieder mit »POKE« zurückgeschrieben. Dabei mußte außerdem noch eine Speicherverschiebung vorgenommen werden, da der Bildschirmspeicher für eine komplette Abbildung des Bodenbildes (640 x 640 Punkte) nicht ausreicht. Diese Prozedur benötigte verhältnismäßig viel Zeit. Das GFA-Basic bietet für derartige Speicherverknüpfungen komfortable Befehle. Wir entschieden uns für »RC_COPY« (Kopieren von rechteckigen Bildausschnitten innerhalb verschiedener Bildschirmseiten) und »BMOVE« (Speicherverschiebung).

BMOVE quelle,ziel,anzahl
RC_COPY quelle,x,y,breite,hoehe TO ziel,z_x,z_y,modus

Informationen zu »RC_COPY« finden Sie auch im Kapitel 7. Durch die Verwendung dieser Befehle verringerte sich diese Bildvorbereitung auf etwa 1,4 Sekunden. Das ist im Vergleich zur ursprünglich benötigten Zeit wirklich sehr gering.

Alles in allem blieben aber trotz dieser Verbesserungen natürlich nach wie vor über zehn Stunden Berechnungszeit für ein einziges Bild. Eine wesentliche Verbesserung konnte nur Punkt 5 unserer Liste (die eigentliche Raytrace-Routine) bringen. Bei näherer Betrachtung erwiesen sich diese Programmteile als sehr umfangreiche Berechnungen. Jeder Bildpunkt wird dabei als »Sehstrahl« in den imaginären Raum verfolgt, um Schnittpunkte zu anderen Objekten und Seitenwänden zu ermitteln. Hierzu werden viele Variablen benötigt sowie etliche Verzweigungsbefehle abgearbeitet.

Der nächste Ansatzpunkt für unsere Programmverbesserung waren die besagten Variablen, wir kommen gleich noch darauf zurück. Zum Schluß konnten durch günstigere Anordnungen der IF-THEN-Entscheidungen (z.B. Verwendung von ELSE-IF) einige Prozent Geschwindigkeitsgewinn erreicht werden. Es wurden in den verschachtelten Schleifen einfach weniger Abfragen durchlaufen, was natürlich eine schnellere Abarbeitung bedeutet. Dazu muß man sich natürlich im klaren sein, welche Abfrage in verschachtelten Schleifen besonders häufig durchlaufen wird.

Der Berechnungsalgorithmus selbst konnte nicht geändert werden. Es wird hier schon eine besondere Variante der Raytrace-Berechnung angewandt, die davon ausgeht, daß es in einem imaginären Raum nur an den Stellen Reflektionen geben kann, an denen sich Objekte (hier spiegelnde Kugeln) befinden. Somit kann die Strahlverfolgung im übrigen Bereich schneller erfolgen, um es einfach zu nennen, »ungenauer«. In unserem Programm wird deshalb in der Trace-Routine überprüft, ob der Strahl eine sogenannte »Kugelbox« erreicht hat, um dann in der Prozedur »inbox« weiterzurechnen. Hier wird dann die Reflexion des Strahls und seine neue Richtung berechnet.

Die hauptsächlichen »Zeitschlucker« sind jedoch, so paradox es klingt, die Variablen bzw. die Berechnungen mit den Variablen vom Typ Fließkomma. Im Programm wurden also hauptsächlich diese Fließkommawerte genutzt, was auch eigentlich notwendig war, um die kleinsten Werte, die auftreten können (1×10^{-3}), darzustellen. Eine altbekannte Sache ist allerdings, daß Fließkommaberechnungen im Vergleich zu Integerberechnungen (Ganzzahlberechnungen) erheblich mehr Aufwand und damit mehr Zeit für den Basic-Interpreter oder -Compiler und somit auch für den Mikroprozessors bedeuten. Wer sich schon einmal mit der Maschinensprache des im Atari ST verwendeten Mikroprozessors vom Typ 68000 (kurz 68K) befaßt hat, weiß, daß es möglich ist, mit nur einem Maschinensprachebefehl eine 16-Bit-Zahl zu multiplizieren oder zu dividieren. Fließkommaberechnungen benötigen dagegen ungleich größeren Programmaufwand, was sich in der Berechnungszeit niederschlägt. Der Vorteil des vergleichsweise riesigen Zahlenbereiches der Fließkommazahlen geht natürlich bei den Intergerwerten (hier maximal 32 Bit) gänzlich unter.

Aber vielleicht ist es ja doch möglich, im Raytrace-Programm Integerwerte zu nutzen. Das sagten wir uns auch, und, um es vorwegzunehmen, unsere Bemühungen waren auch sehr erfolgreich. Die oben angesprochene 10-Stunden-Berechnung (Bild 6.8) dauerte nun nur noch etwa 136 Minuten, also knapp über zwei Stunden! Nebenbei bemerkt war jetzt unser Interpreterprogramm sogar schneller als das Ursprungsprogramm (in Omikron-Basic) in kompilierter Form.

Im Programm wird jetzt eine Methode angewandt, die auch für Sie und Ihre weiteren Programmprojekte sehr interessant werden kann. Es handelt sich um die Verwendung der sogenannten normalisierten Zahlen. Dabei werden reelle Zahlen als Integerwert und als »Normalisierungsfaktor« dargestellt:

reelle Zahl = Integerwert / Normalisierungsfaktor

Je nach Größe des Faktors (vorteilhafterweise eine Potenz von 2) und der maximalen Größe des Integerwertes kann damit die Genauigkeit der Zahl bestimmt werden. Mit diesen Zahlen kann nun genauso gerechnet werden wie mit den Fließkommazahlen. In unserem Programm haben wir das auch gemacht, wobei allerdings der Faktor meist schon in den Integerwerten eingerechnet wurde, um unnötige (und zeitaufwendige) Multiplikationen oder Divisionen zu vermeiden. Weiter möchten wir an dieser Stelle nicht auf dieses spezielle Gebiet eingehen. Sie sehen an unserem Programm sehr gut, wie man Berechnungen fast ausschließlich durch Integerwerte machen kann.

Bevor die Berechnungsroutinen auf Integer umgestellt werden, muß man sich genau darüber im klaren sein, welche maximalen und minimalen Ergebnisse erreicht werden. Somit ist es unvermeidlich, das komplette Programm auf diese Bedingungen zu überprüfen. Die Maximalwerte für Integervariablen sind im Gegensatz zu Fließkommavariablen vergleichsweise sehr klein. Wird der Bereich überschritten (gesprengt), erhalten Sie eine Fehlermeldung. Wenn Sie dagegen die erlaubten Bereiche einhalten, werden Sie mit hoher Rechengeschwindigkeit belohnt. Weiterhin ist bei der Geschwindigkeitsoptimierung zu beachten, ob es nicht sinnvoller (und schneller) ist, Konstantwerte als Fließkommawerte anzugeben, statt der Verwendung eines Integers und eines Faktors. In unserem Programm sehen Sie das zum Beispiel in der Funktion »inbox«. Wir arbeiten dort mit einer Variablen »rsqr()«, dem Quadrat der einzelnen Kugeldurchmesser. Hier wird der maximale Darstellungsbereich der 32-Bit-Werte sehr schnell überschritten. Man müßte also die Werte als Produkt aus integer x Faktor angeben, was aber eine zusätzliche Muliplikation bedeutet und damit zusätzlichen Zeitaufwand. Unsere Versuche zeigten, daß es in diesem Fall günstiger ist, die Fließkommavariable zu nutzen. Genauso verhält es sich mit der Variablen »dot«, ebenfalls in der Funktion »inbox«.

Zur weiteren Geschwindigkeitsoptimierung zählen die *Integerfunktionen* bei Multiplikation, Division, Addition und Subtraktion. So ist zum Beispiel die folgende erste Schreibweise günstiger als die zweite:

```
DIV(x%,y%)
x%=x%/y%
```

Unsere Geschwindigkeitsoptimierungen bringen im Raytrace-Programm auch einen nicht zu übersehenden Nachteil mit sich. So ist es mitunter möglich, daß kleine Kugeln (Durchmesser unter 50 Punkten) einfach »übersehen« werden. Das liegt am sogenannten Stepwert, der die Geschwindigkeit der Strahlverfolgung festlegt. Sie finden die dazugehörige Variable zu Programmbeginn als »stepgross%« mit dem Wert 100.000.000. Ein größerer Wert bedeutet schnellere, aber auch ungenauere Bearbeitung. Probieren Sie das doch einmal aus, indem Sie hier 200 Millionen oder noch größere Werte einsetzen. Umgekehrt kann ein kleinerer Wert genauere Berechnungen bringen, allerdings auf Kosten der Zeit. Sollen also auch kleinere Kugeln dargestellt werden, sollte dieser Wert ebenfalls verkleinert werden (z.B. 10 Millionen).

Die maximale Anzahl von Objekten (Kugeln) ist in unserem Programm auf 10 festgelegt, was Sie bei Bedarf natürlich auch ändern können. Ob der maximal dastellbare Raum von 640 x 400 x 640 Punkten für mehr als 10 Kugeln ausreicht, ist allerdings fraglich. Wir haben in unseren Beispielbildern maximal drei Kugeln definiert.

Sollten Sie einmal nur eine Abbildung wie in Bild 6.6 rechts benötigen, können Sie ebenfalls dieses Programm nutzen und dazu weiter beschleunigen, indem Sie in der Funktion »trace« die FOR-NEXT-Schleife »FOR i%=1 TO numobs%« ganz herausnehmen. In der Prozedur »bildschirm_vorbereiten« können Sie verhindern, daß die Schatten gezeichnet werden, obwohl in der Datendatei zum Programm 6_2.GFA mindestens eine Kugel angegeben werden muß.

Soweit zu den Optimierungen unseres Programms 6_2.GFA. Experimentieren Sie doch selbst einmal mit weiteren Beschleunigungsmöglichkeiten. So können zum Beispiel Integermultiplikationen oder Divisionen auch durch bitweises Verschieben der Werte erreicht werden. Verschieben um 1 Bit nach rechts bedeutet eine Division durch 2, das Verschieben um 1 Bit nach links dagegen eine Multiplikation mit 2. Versuchen Sie auch einmal, den Stepwert als Potenz zu 2 anzugeben und dann mit den Bitverschiebungen zu arbeiten.

Wer möchte, kann das Programm selbstverständlich auch auf Farbbetrieb umstellen. Die Berechnung wäre aufgrund der geringeren Auflösung sogar noch schneller.

Es folgt nun endlich das komplette Listing.

```
' ************************ PROGRAMM 6.2 ************************
' *************************************************************
' *****                                                   *****
' *****                    RAYTRACING                     *****
' *****      PROGRAMM ZUR BERECHNUNG VON LICHTSTRAHLEN    *****
' *****                                                   *****
' *****         IDEE UND ALGORITHMEN VON TOM HUDSON       *****
' *****                                                   *****
' *****      VERSION IN GFA-BASIC VON WILHELM BESENTHAL   *****
' *****                  UND  JENS MUUS                   *****
' *****                                                   *****
' *****            MARKT & TECHNIK, MÜNCHEN 1989          *****
' *****                                                   *****
' *************************************************************
'
'
vorbereitungen                           !Speicherreservierung usw.
werte_lesen                              !Datenfile und Bilder lesen
stepgross%=100000000                     !Anfangswert für
                                         !Strahlberechnung
'
kugelstil=2                              !Füllstil für Kugelschatten
kugelmuster=8
brettstil=2                              !Füllstil für Schachbrettmuster
brettmuster=8
bildschirm_vorbereiten                   !Kugeln und Schatten zeichnen
ON BREAK GOSUB ende
'
timer_anfang=TIMER                       !Für die berechnung der
                                         !Ausführungszeit
FOR iy%=1000 TO 400000 STEP 1000         !Beginn der Tracing-Routine
  FOR ix%=1000 TO 640000 STEP 1000       !400 Zeilen und 640 Spalten
    bounct%=0
    IF FN trace
      SLPOKE e%,bild%
      PSET DIV(ix%,1000),SUB(400,DIV(iy%,1000)),1
    ENDIF
  NEXT ix%
  IF MOUSEK>0
    break&=1                             !Vorzeitiger Abbruch
    iy%=401000
  ENDIF
```

```
NEXT iy%                                    !Ende der Tracing-Routine
sekunden=(TIMER-timer_anfang)/200           !Ausführungszeit
'
SLPOKE e%,bild%                             !Ursprünglichen Zustand
                                            !wiederherstellen
IF break&
  ALERT 2,"Vorzeitiger Programmabbruch.|Bild abspeichern?",2,"JA|NEIN",var
  IF var=2
    ende
  ENDIF
ENDIF
BSAVE outfile$,XBIOS(2),32000               !Fertiges Bild abspeichern
ALERT 0,"Die Berechnung dauerte|"+STR$(sekunden)+
        " Sekunden.",1,"ENDE",var
ende
```

Bevor die Routine »bildschirm_vorbereiten« aufgerufen wird, werden die Werte für die Füllmuster für die Kugelschatten und die Schachbrettmuster angegeben. Das Schachbrettmuster wird nur für den Boden des Ausgabebildes benötigt. Das Muster der Decke wird während des Raytracings berechnet. Die Kugelschatten werden durch eine ODER-Verknüpfung zwischen Kugelfüllmuster und Boden errechnet. Wir haben hier eine vollständig schwarze Fläche als Füllmuster voreingestellt. In Bild 6.8 sehen Sie, daß auch andere Muster gut aussehen können.

Das Programm kann jederzeit durch [Control]+[Shift]+[Alternate] abgebrochen werden. Es wird dann die Prozedur »ende« aufgerufen, die einen ordnungsgemäßen Programmabbruch garantiert. Wenn Sie die Berechnung abbrechen möchten und das unvollständige Bild abspeichern möchten, können Sie dies mit der rechten Maustaste erreichen. Allerdings wird die Taste jeweils nur am Ende der Zeilen abgefragt. Halten Sie sie also so lange gedrückt, bis eine Alert-Box erscheint.

Es folgen die Unterprogramme und Funktionen. Hier ist zunächst die eigentliche Trace-Funktion, die jeweils einen Strahl verfolgt und die Farbe des Zielpunktes zurückliefert. Dazu wird die Anfangsadresse des entsprechenden Grafikbildes in eine Systemvariable abgelegt und anschließend der Befehl »POINT« genutzt.

Die Trace-Funktion besteht genaugenommen aus zwei Funktionen: eine für die grobe (ungenaue) und eine für die genaue Berechnung (sie heißt inbox). Dies ist eine Methode, die leider erheblichen Berechnungszeiten zu verringern. Dabei geht man davon aus, daß außerhalb der Bereiche, in denen sich Objekte (Kugeln) befinden, keine Reflexion stattfindet, sondern einfach nur die Farbe der Wand ermittelt werden muß. In den Routinen »trace« und »inbox« sehen Sie »ellenlange« verschachtelte IF-THEN-Schleifen. Diese hätte man auch sehr viel komprimierter darstellen können (durch Verknüpfungen, wie z.B. AND oder die Verwendung von ELSE IF), dadurch wäre die Berechnung aber um einiges länger geworden. Häufig verwenden wir Long-Integer-Variablen in Berechnungen, obwohl

Integer-Word ausgereicht hätten, aber auch hierdurch wäre das Programm langsamer geworden.

```
FUNCTION trace                              !Tracingroutine für einen Punkt
  xi%=((ix%-viewxyz%(0)))/(-viewxyz%(2))*1000
  yi%=((iy%-viewxyz%(1)))/(-viewxyz%(2))*1000
  zi%=1000
  grosstep
  rayx%=ix%                                 !Werte übernehmen
  rayy%=iy%
  rayz%=10000
  lastobj%=99
  follow:                                   !Funktion Inbox, wenn der Strahl
  IF rayx%<1
  ELSE IF rayx%>639000
  ELSE IF rayy%<1
  ELSE IF rayy%>399000
  ELSE IF rayz%<1
  ELSE IF rayz%<640000
    px%=rayx%
    py%=rayy%
    pz%=rayz%
    ADD rayx%,xi%                           !Strahlkoordinaten
    ADD rayy%,yi%
    ADD rayz%,zi%
    FOR i%=1 TO numobs%                     !Innerhalb einer Kugelbox?
      IF i%=lastobj%
      ELSE IF rayx%<minx%(i%)
      ELSE IF rayx%>maxx%(i%)
      ELSE IF rayy%<miny%(i%)
      ELSE IF rayy%>maxy%(i%)
      ELSE IF rayz%<minz%(i%)
      ELSE IF rayz%>maxz%(i%)
      ELSE IF FN inbox<0
        RETURN 1
      ENDIF
    NEXT i%
    GOTO follow
  ENDIF
  FOR i%=1 TO 8                             !Strahl hat den Raum verlassen
    done%=1
    midx%=ADD(px%,rayx%)/2
    midy%=ADD(py%,rayy%)/2
    midz%=ADD(pz%,rayz%)/2
```

```
  IF midx%<640000
    IF midx%>1
      IF midy%<400000
        IF midy%>1
          IF midz%<640000
            IF midz%>1
              px%=midx%
              py%=midy%
              pz%=midz%
              done%=0
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
  IF done%
    rayx%=midx%
    rayy%=midy%
    rayz%=midz%
  ENDIF
NEXT i%
tfx%=MIN(MAX(0,midx%/1000),639)              !Kollisionspunkt in das
tfy%=MIN(MAX(0,midy%/1000),399)              !Innere des Raumes verlegen
tfz%=MIN(MAX(0,midz%/1000),639)
'
IF tfx%=639
  SLPOKE e%,rbuf%
  RETURN PTST(SUB(639,tfz%),SUB(399,tfy%)) !Rechte Wand
ENDIF
IF tfx%=0
  SLPOKE e%,lbuf%
  RETURN PTST(tfz%,SUB(399,tfy%))          !Linke Wand
ENDIF
IF tfy%=0
  IF tfz%>399
    SLPOKE e%,botbuf2%
    RETURN PTST(tfx%,tfz%-400)
  ELSE
    SLPOKE e%,botbuf%
    RETURN PTST(tfx%,tfz%)
  ENDIF
ENDIF
```

```
  IF tfy%=399
    RETURN (tfx%\40+tfz%\40) AND 1            !Decke
  ENDIF
  IF tfz%=0
    SLPOKE e%,fbuf%
    RETURN PTST(SUB(639,tfx%),SUB(399,tfy%)) !Vordere Wand
  ENDIF
  SLPOKE e%,bbuf%
  RETURN PTST(tfx%,SUB(399,tfy%))            !Rückwand
ENDFUNC
```

Es folgt die Funktion »inbox«, die den Strahlverlauf innerhalb einer sogenannten Kugelbox berechnet. Der Vorteil dieser Methode ist, daß außerhalb der Kugelbox ungenauer, das heißt schneller, berechnet werden kann.

```
FUNCTION inbox !GENAUERE STRAHLVERFOLGUNG INNERHALB EINER KUGELBOX
  DIV xi%,2                                  !Stepwert verkleinern auf 1
  DIV yi%,2
  DIV zi%,2
  boxloop:                                   !Strahlberechnung innerhalb einer
                                             !Kugelbox
  CLR sphtot%,boxtot%
  FOR i%=1 TO numobs%
    insphere%(i%)=0
    inbox%(i%)=0
    IF rayx%>=minx%(i%)
      IF rayx%<=maxx%(i%)
        IF rayy%>=miny%(i%)
          IF rayy%<=maxy%(i%)
            IF rayz%>=minz%(i%)
              IF rayz%<=maxz%(i%)
                inbox%(i%)=1
                INC boxtot%
                IF SUB(rayx%,xs%(i%))^2+SUB(rayy%,ys%(i%))
                       ^2+SUB(rayz%,zs%(i%)) ^2-rsqr(i%)<0
                  insphere%(i%)=1
                  INC sphtot%
                  sphnum%=i%
                ENDIF
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
```

```
    ENDIF
  NEXT i%
  IF boxtot%=0
    grosstep
    RETURN 1                          !Der Strahl hat die Box verlassen
  ENDIF
  IF sphtot%>0                        !Kollision ermitteln und Strahl reflektieren
    IF bounct%>=reflects%-1
      RETURN -1
    ENDIF
    FOR j%=1 TO 8                     !Objekt suchen
      testx%=SHR(ADD(px%,rayx%),1)
      testy%=SHR(ADD(py%,rayy%),1)
      testz%=SHR(ADD(pz%,rayz%),1)
      CLR sphtot%                     !Zähler zurücksetzen
      FOR i%=1 TO numobs%
        IF inbox%(i%)
          IF SUB(testx%,xs%(i%))^2+SUB(testy%,ys%(i%))
                 ^2+SUB(testz%,zs%(i%))^2-rsqr(i%)<0
            insphere%(i%)=1
            INC sphtot%
            sphnum%=i%
          ENDIF
        ENDIF
      NEXT i%
      IF sphtot%=0
        px%=testx%
        py%=testy%
        pz%=testz%
      ELSE
        rayx%=testx%
        rayy%=testy%
        rayz%=testz%
      ENDIF
    NEXT j%
    ' -----------------------           REFLEKTIERTEN STRAHL WEITERBERECHNEN
    lastobj%=sphnum%                    !Nummer des getroffenen Objekts
    INC bounct%
    nx%=SUB(rayx%,xs%(sphnum%))
    ny%=SUB(rayy%,ys%(sphnum%))
    nz%=SUB(rayz%,zs%(sphnum%))
    dot=ABS(ADD(ADD(MUL(xi%,nx%),MUL(yi%,ny%)),MUL(zi%,nz%)))
    xi%=ADD(DIV(xi%,dot),MUL(nx%,2))
```

```
    yi%=ADD(DIV(yi%,dot),MUL(ny%,2))
    zi%=ADD(DIV(zi%,dot),MUL(nz%,2))
    grosstep                                !Großen Stepwert einstellen
  ELSE                                      !Noch einmal genaue Berechnung
    ADD rayx%,xi%
    ADD rayy%,yi%
    ADD rayz%,zi%
    GOTO boxloop
  ENDIF
  RETURN 1
ENDFUNC
'
PROCEDURE grosstep
  mfac%=stepgross%/MAX(ABS(xi%),MAX(ABS(yi%),ABS(zi%)))
  xi%=xi%*mfac%/1000
  yi%=yi%*mfac%/1000
  zi%=zi%*mfac%/1000
RETURN
```

Das Programm benötigt einige wichtige Daten, die aus einem ASCII-File von Diskette eingelesen werden. Wir haben als Extension für diese Dateien ».CTL« eingesetzt. In der Datei sind die Daten zeilenweise eingetragen. Es ist auch erlaubt, Bemerkungen dort einzutragen. Um diese eindeutig zu kennzeichnen, wird ihnen ein Semikolon zu Beginn der Zeile vorangestellt.

In der Datei werden die Informationen in folgender Reihenfolge erwartet:

1. Maximale Anzahl von Reflexionen, die berechnet werden sollen.
2. »Viewpoint«, der Betrachtungspunkt, der den Blickwinkel des Bildes bestimmt. Dabei werden die Daten in der Reihenfolge X (1 bis 640), Y (1 bis 400) und Z (–1 bis –640) erwartet. Für Z muß ein negativer Wert angegeben werden, da der Nullwert auf der Z-Achse ja der Bildschirm ist. »In die Tiefe« wird der Wert positiv und zum Betrachter negativ.
3. Anzahl der Objekte (Kugeln), Werte von 1 bis 10.
4. Objektgröße (bei mehr als einem Objekt müssen die Werte hintereinander durch Kommas getrennt angegeben werden).
5. X-Position(en) der Objekte.
6. Y-Position(en) der Objekte.
7. Z-Position(en) der Objekte.

8. Vollständiger Dateiname (vollständiger Pfadname) der Bilddatei für die linke Wand des Raumes. Diese Bilddatei muß natürlich ebenfalls auf der Diskette vorhanden sein. Dabei besteht diese lediglich aus den 32.000 Byte Bilddaten. So ein Bild kann zum Beispiel unter GFA-Basic mit »BSAVE« abgespeichert werden.

9. Wie 8, jedoch Name für Bilddatei für die rechte Wand.

10. Wie 8, jedoch für die hintere Wand.

11. Wie 8, jedoch für die Front. Diese ist auf dem fertigen Bild natürlich eventuell nur als Spiegelung sichtbar, da unser Bildschirm ja das fertige Bild annimmt.

12. Hier muß der vollständige Dateiname für das fertige Bild angegeben werden. Dieser Name sollte sehr sorgfältig angegeben werden, da das Programm eventuelle Fehler erst nach der Berechnung feststellen würde. Wenn bis dahin mitunter mehrere Stunden vergangen sind und das Programm dann mit einer Fehlermeldung abbricht, würden Sie sich bestimmt sehr ärgern! Achten Sie also darauf, daß der Dateiname einwandfrei und auch noch genügend Platz auf der Diskette vorhanden ist!

13. Um das Ende der Parameterdatei zu kennzeichnen, geben Sie hier das Wort »end« ein.

Auf der beiliegenden Diskette finden Sie im Ordner 6.GFA eine Beispieldatei, die wir nutzten, um das Bild 6.9 zu berechnen (Datei RAYTRACE.CTL). Es folgen die übrigen Routinen des Programms. Zunächst liest »werte_lesen« die Datendatei und wertet die Parameter aus. Die Prozedur ist recht aufwendig, erkennt dafür aber auch fast alle möglichen fehlerhaften Angaben in der Datei.

```
PROCEDURE werte_lesen                       !Daten aus dem Datenfile lesen
  IF EXIST(filename$)
    OPEN "I",#1,filename$
    @daten_lesen
    reflects%=VAL(text$)                    !Anzahl der Reflektionen
    IF reflects%>10 OR reflects%<1
      PRINT "Falscher Reflektionswert ";reflects%;"
            (Richtig: 1 bis 10)!"
      abbruch
    ENDIF
    @daten_lesen
    s$=text$+","                            !Betrachtungswinkel
    p=INSTR(1,s$,",")
    IF p=-1
      PRINT "Falsche Angaben für den Betrachtungswinkel!"
      abbruch
    ENDIF
    viewxyz%(0)=VAL(LEFT$(s$,p))*1000
    s$=RIGHT$(s$,LEN(s$)-p)
```

```
p=INSTR(1,s$,",")
IF p=-1 THEN
  PRINT "Falsche Angaben für den Betrachtungswinkel!"
  abbruch
ENDIF
viewxyz%(1)=VAL(LEFT$(s$,p))*1000
s$=RIGHT$(s$,LEN(s$)-p)
p=INSTR(1,s$,",")
IF p=-1 THEN
  PRINT "Falsche Angaben für den Betrachtungswinkel!"
  abbruch
ENDIF
viewxyz%(2)=VAL(LEFT$(s$,p))*1000
s$=RIGHT$(s$,LEN(s$)-p)
IF viewxyz%(2)>=0
  PRINT "Falsche Angaben für den Betrachtungswinkel (Z>=0)!"
  abbruch
ENDIF
@daten_lesen
numobs%=VAL(text$)                    !Anzahl der Objekte (Kugeln)
IF numobs%>20 OR numobs%<0
  PRINT "Anzahl der Objekte ist falsch (Richtig: 1 bis 20)!"
  abbruch
ENDIF
@daten_lesen
s$=text$+","                          !Größe der einzelnen Objekte einlesen
FOR i%=1 TO numobs%
  p=INSTR(1,s$,",")
  IF p=-1
    PRINT "Falscher Wert (Objektgröße)."
    abbruch
  ENDIF
  sizes%(i%)=VAL(LEFT$(s$,p))*1000
  s$=RIGHT$(s$,LEN(s$)-p)
  IF sizes%(i%)<30000 OR sizes%(i%)>199000
    PRINT "Falscher Wert für Objektgröße
          (Richtig: 30 bis 199)!"
    abbruch
  ENDIF
  rsqr(i%)=sizes%(i%)^2
NEXT i%
@daten_lesen
s$=text$+","                          !Alle X-Positionen der Objekte einlesen
```

```
FOR i%=1 TO numobs%
  p=INSTR(1,s$,",")
  IF p=-1
    PRINT "Falsche X-Koordinate (Objektkoordinate)."
    abbruch
  ENDIF
  xs%(i%)=VAL(LEFT$(s$,p))*1000
  s$=RIGHT$(s$,LEN(s$)-p)
  minx%(i%)=xs%(i%)-sizes%(i%)
  maxx%(i%)=xs%(i%)+sizes%(i%)
  IF minx%(i%)<1000 OR maxx%(i%)>638000
    PRINT "Falsche X-Koordinate (Objektkoordinate)."
    abbruch
  ENDIF
NEXT i%
@daten_lesen
s$=text$+","                         !Alle Y-Positionen der Objekte einlesen
FOR i%=1 TO numobs%
  p=INSTR(1,s$,",")
  IF p=-1
    PRINT "Falsche Y-Koordinate (Objektkoordinate)."
    abbruch
  ENDIF
  ys%(i%)=VAL(LEFT$(s$,p))*1000
  s$=RIGHT$(s$,LEN(s$)-p)
  miny%(i%)=ys%(i%)-sizes%(i%)
  maxy%(i%)=ys%(i%)+sizes%(i%)
  IF miny%(i%)<1000 OR maxy%(i%)>398000
    PRINT "Falsche Y-Koordinate (Objektkoordinate)."
    abbruch
  ENDIF
NEXT i%
@daten_lesen
s$=text$+","                         !Alle Z-Positionen der Objekte einlesen
FOR i%=1 TO numobs%
  p=INSTR(1,s$,",")
  IF p=-1
    PRINT "Falsche Z-Koordinate (Objektkoordinate)."
    abbruch
  ENDIF
  zs%(i%)=VAL(LEFT$(s$,p))*1000
  s$=RIGHT$(s$,LEN(s$)-p)
  minz%(i%)=zs%(i%)-sizes%(i%)
```

```
      maxz%(i%)=zs%(i%)+sizes%(i%)
      IF minz%(i%)<1000 OR maxz%(i%)>638000
        PRINT "Falsche Z-Koordinate (Objektkoordinate)."
        abbruch
      ENDIF
    NEXT i%
    @daten_lesen
    lfile$=text$                    !Dateiname der Bilddatei für linke Wand
    @daten_lesen
    rfile$=text$                    !Dateiname der Bilddatei rechte Wand
    @daten_lesen
    bfile$=text$                    !Dateiname der Bilddatei Boden
    @daten_lesen
    ffile$=text$                    !Dateiname der Bilddatei Vorderseite
    @daten_lesen
    outfile$=text$                  !Dateiname für das fertige Bild - sehr wichtig!
    CLOSE #1
    BLOAD lfile$,lbuf%              !Die Wandbilder einladen
    BLOAD rfile$,rbuf%
    BLOAD bfile$,bbuf%
    BLOAD ffile$,fbuf%
  ELSE
    ende
  ENDIF
RETURN
'
PROCEDURE daten_lesen               !Eine Datenzeile lesen
  text$=";"
  WHILE LEFT$(text$,1)=";"
    IF EOF(#1)                      !Zu wenig Daten
      PRINT "Die ENDE-Kenzeichnung fehlt!"
      abbruch
    ENDIF
    LINE INPUT #1,text$
    IF LEFT$(text$,3)="END" OR LEFT$(text$,3)="end" OR EOF(#1)
      PRINT "Zu wenig Daten!"
      abbruch                       !Zu wenig Daten
    ENDIF
    IF LEN(text$)>50
      PRINT "Die Datenzeile ist zu lang!"
      abbruch
    ENDIF
```

```
  WEND
RETURN
'
PROCEDURE vorbereitungen                     !Variablen usw.
  RESERVE FRE(0)-200192                      !Speicher für die Bildschirme
  DIM sizes%(20),xs%(20),ys%(20),zs%(20),rsqr(20)
  DIM minx%(20),maxx%(20),miny%(20),maxy%(20),minz%(20),maxz%(20)
  DIM inbox%(20),insphere%(20),viewxyz%(20)
  ' -------------------------              ANFANGSADRESSEN DER BILDER ERMITTELN
  speicherad=MALLOC(180480)
  botbuf%=speicherad AND &HFFFFF00           !51200 Bytes für den Boden
  botbuf2%=botbuf%+32000
  lbuf%=botbuf%+51200                        !Je Seite eine Bildschirmseite
  rbuf%=lbuf%+32256
  bbuf%=rbuf%+32256
  fbuf%=bbuf%+32256
  e%=&H44E                                   !Enthält Adresse des Bildschirm-RAMs
  bild%=LPEEK(e%)
  IF XBIOS(4)<>2
    ALERT 1,"Nur Hires",1,"ENDE",var
    ende
  ENDIF
  workpath$=CHR$(GEMDOS(25)+65)+":"+DIR$(0)+"\6.GFA\"  !PFAD ERM.
  FILESELECT workpath$+"*.CTL","",filename$          !Für CTL-File
RETURN
'
PROCEDURE bildschirm_vorbereiten             !Kugeln zeichnen usw.
  CLS
  CLIP 0,0,640,400
  GRAPHMODE 1
  DEFFILL 1,kugelstil,kugelmuster            !Füllstil für Kugeln
  FOR i%=1 TO numobs%                        !Kugeln f. Schattenberechnung zeichnen
    PCIRCLE xs%(i%)/1000,zs%(i%)/1000-400,sizes%(i%)/1000
  NEXT i%
  BMOVE bild%,botbuf%+32000,19200
  CLS
  FOR i%=1 TO numobs%
    PCIRCLE xs%(i%)/1000,zs%(i%)/1000,sizes%(i%)/1000
  NEXT i%
  BMOVE bild%,botbuf%,32000
  CLS
  '
  DEFFILL 1,brettstil,brettmuster            !Füllstil für Schachbrettmuster
```

```
  FOR i%=0 TO 7
    FOR j%=0 TO 15                    !Schachbrettmuster zeichnen
      IF ((i%+j%) AND 1)
        PBOX j%*40,i%*50,j%*40+39,i%*50+49
      ENDIF
    NEXT j%
  NEXT i%
  RC_COPY bild%,0,0,640,400 TO botbuf%,0,0,7
  RC_COPY bild%,0,0,640,240 TO botbuf2%,0,0,7
  CLS
RETURN
'
PROCEDURE abbruch
  ALERT 3,"Vorzeitiger Programmabbruch.",1,"ENDE",var
  ende
RETURN
'
PROCEDURE ende
  SLPOKE e%,bild%                     !Ursprünglichen Zustand wiederherstellen
  CLOSE #1
  ~MFREE(speicherad)
  RESERVE FRE(0)+200192
  END
RETURN
```

Damit ist das Programmlisting beendet. Wir hoffen, daß Ihnen dieses Programm sehenswerte Bilder berechnet. Ein weiteres Beispiel dazu können Sie in Bild 6.9 sehen. Das Programm benötigt übrigens trotz der Geschwindigkeitsoptimierung noch immer relativ viel Zeit für die Berechnung eines Bildes (unkompiliert hier »nur« etwa 2 Stunden). Das Kompilieren dieses Programms bringt natürlich eine weitere erhebliche Zeitersparnis.

In Bild 6.9 erkennen Sie einen kleinen Fehler im Schatten der hinteren Kugel. Das liegt nicht an der Strahlverfolgung unseres Programms, sondern am GFA-Basic (Version 3.04). Wenn nämlich ein gefüllter Kreis mit negativer Y-Koordinate für den Mittelpunkt gezeichnet werden soll, bleibt am Bildschirmrand ein Streifen von zwei Pixeln »ungefüllt« (bei eingeschaltetem Clipping). Genau das passierte auch bei dem Schatten der hinteren Kugel. Dieser befindet sich zwar mitten auf dem Bild des Bodens, er wurde aber teilweise außerhalb des Bildschirms gezeichnet. Und das liegt daran, daß wir ein Bildschirmformat von 640 x 640 Punkten nutzen. Um darin die Figuren (hier die Schatten und das Schachbrettmuster) zeichnen zu können, mußte das Bild in zwei Teilen auf dem Bildschirm dargestellt werden.

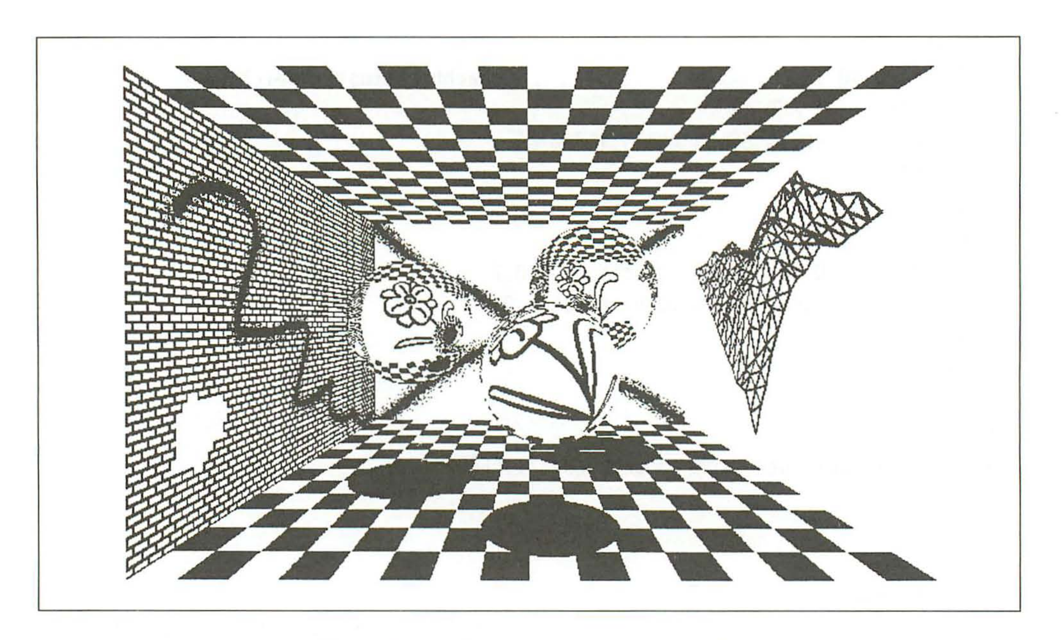


Bild 6.9: Durch Raytracing berechnetes Bild

6.4.2 Fraktale

Das Wort »Fraktal« ist in letzter Zeit zu einem regelrechten Hit aufgestiegen. Man stößt auch in der Computer-Fachliteratur ständig auf diesen Begriff. Natürlich fragen sich die meisten unter Ihnen, was denn nun so ein Fraktal ist?

Ein Fraktal ist ein Gebilde mit unendlich fein gewundener oder zerklüfteter Grenze oder Oberfläche. Doch was hat das denn mit Computern zu tun? Nun, erst einmal nichts, dazu müssen wir zunächst doch noch ein wenig weiter ausholen.

Ein Name ist untrennbar mit den Fraktalen verbunden: *Benoid B. Mandelbrot*. Dieser Mann, ein Physiker, entwarf mathematische Modelle für die sogenannte »Chaos«-Theorie. Dabei handelt es sich um ein sehr aktuelles Forschungsgebiet. Die Chaos-Theorie besagt, daß alles auf dieser Welt, was sich uns Menschen scheinbar geregelt bzw. regelmäßig darstellt, eigentlich gar nicht so regelmäßig ist. Man ist schon im Altertum von der falschen Annahme ausgegangen, daß alle Dinge auf der Welt in einfache geometrische Figuren passen (Berg – Kegel, Wolke – Kreis oder Oval). Zweifellos ist dieser Vergleich der Natur mit von Menschen geschaffenen Objekten falsch, obwohl auch heute immer wieder derartige Beschreibungen erscheinen. Gegenbeispiele dazu findet man überall. Das beste ist sicherlich das Wetter. Das einzig regelmäßige am Wetter ist für uns, daß es bei uns im Sommer wärmer ist als im Winter! Kein Meteorologe kann mit Sicherheit sagen, wie das Wetter am folgenden Tage oder selbst in einer Stunde aussieht. Das Wettergeschehen ist einfach für jede Berechnung zu komplex. Die Forscher fanden außerdem heraus, daß schon

kleinste Einflüsse in einem Geschehen wie zum Beispiel dem Wetter riesige, ja sogar katastrophale Auswirkungen haben können. Somit kann theoretisch »der Flügelschlag eines Schmetterlings« einen Wetterumschwung bewirken.

Ein einfacheres Beispiel: Betrachten Sie einmal einen tropfenden Wasserhahn. Sie werden sehen, daß dort scheinbar sehr regelmäßig Tropfen herabfallen. Beobachtet man diese Szenerie einmal mit einer Hochgeschwindigkeitskamera, wird man feststellen, daß die Tropfen keineswegs »regelmäßig« fallen, sondern daß es große Tropfen, kleine Tropfen, zerplatzende Tropfen gibt. Auch hier kann man das Beobachtete kurz als »Durcheinander« oder »Chaos« bezeichnen. Genau wie das Chaos beim Wetter!

Diese Entdeckung machten auch die Forscher, die daraus schlossen, daß das Durcheinander immer an Übergangsbereichen zwischen festen Zuständen ansteht. Auf den Wasserhahn bezogen sind die »festen« Zustände ein ganz geschlossener und ein ganz geöffneter Wasserhahn. Alles, was dazwischen ist, ist das besagte Durcheinander. Und das ist auch mit den besten Mitteln nicht vorhersehbar.

Das Durcheinander in den *Übergangsbereichen* kann man tatsächlich überall beobachten. So werden zum Beispiel Eskalationen, die zu Kriegen führen, ebenso als chaotischer Zustand im Übergangsbereich zwischen festen Zuständen angesehen, wie beispielsweise das Geschehen an der Börse, das plötzlich unberechenbar werden kann.

Man wird vermutlich niemals eine verläßliche Wettervorhersage bekommen, denn das Chaos »Wetter« im Übergangsbereich zwischen der Erde und dem Vakuum des Weltalls ist für jede Berechnung zu komplex.

Nun geht es darum, zu erforschen, wann ein fester Zustand in einen Übergangsbereich übergeht, was dann dabei geschieht, und mit welchen minimalen Einflüssen man in einem Chaos eine gezielte Wirkung hervorruft. Die Wissenschaft zu diesem Thema steckt noch in den Kinderschuhen. Sollten aber eines Tages konkrete Ergebnisse anstehen, könnte das die Welt zweifellos revolutionieren.

Mandelbrot und andere Forscher entdeckten Möglichkeiten, das Chaos in einem Beispiel mittels mathematischer Modelle sichtbar zu machen. Das bekannteste Modell ist das »Apfelmännchen« (siehe auch Bild 6.10), das sicher jeder von Ihnen kennt. Das Apfelmännchen ist eine Figur, die ihren Namen nach dem Äußeren, das einem Apfel mit Armen und Beinen ähnelt, erhielt.

Ein *Fraktal*, und darum handelt es sich bei dem Apfelmännchen, hat also eine unendlich fein zerklüftete Oberfläche. Zu dieser Eigenschaft kommt noch hinzu, daß das Gebilde in sich eine sogenannte Selbstähnlichkeit besitzt, eine ganz besondere Eigenschaft der Fraktale. Das heißt, im Gebilde sind Formen sichtbar, die dem Äußeren desselben Gebildes ähneln. Zum Beispiel werden in einem Apfelmännchen-Bild immer wieder kleinere Apfelmännchen sichtbar.

Mandelbrot definierte also die Formel, die es ermöglichte, das Chaos anhand von Zahlen sichtbar zu machen. Dabei handelt es sich um einen bestimmten Zahlenbereich, nach

seinem Entdecker Mandelbrot-Menge genannt, der mittels der Formel Ergebnisse bringt, die wiederum bei neuen Berechnungen mit derselben Formel hinzugezogen werden. Diesen Vorgang bezeichnet man auch als Rekursion oder Iteration. Mittlerweile gibt es viele solcher mathematischen Modelle. Wir wollen im folgenden Abschnitt jedoch lediglich auf die Mandelbrotsche Definition eingehen.

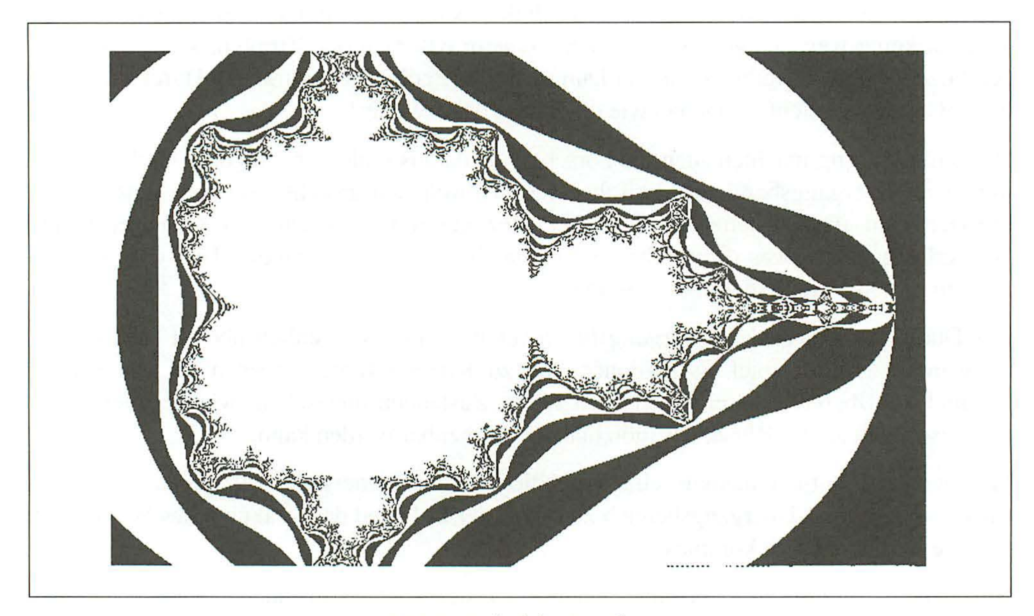


Bild 6.10: Apfelmännchen

Das folgende Programm, auf der Diskette das Programm 6_3.GFA, berechnet die besagten Fraktale. Das dabei entstehende Grundbild sehen Sie in Bild 6.11. Zu dieser Thematik gibt es mittlerweile schon eine große Anzahl Programme, die mehr oder weniger bunte Grafiken auf den Bildschirm zaubern. Dabei wird folgende Grundformel angewandt:

Ergebnis(n+1)=Ergebnis(n) x Ergebnis(n) + komplexe Zahl

Eine derartige komplexe Zahl ist eine Zahl, die aus zwei Teilen besteht, nämlich dem Imaginärteil, der auf dem Bildschirm in der Y-Achse sichtbar ist, und dem Realteil (X-Achse). Es kann dadurch also ein Teilbereich der komplexen Zahlen auf dem Bildschirm dargestellt werden. Bei den Berechnungen interessiert nun nicht das Ergebnis der oben angegebenen Formel, sondern die Anzahl der wiederholten Berechnungen (Iterationen), bis ein bestimmtes Mindest- oder Höchstergebnis erreicht wird. Die Anzahl der Berechnungen wird dann als spezielle Farbe auf dem Bildschirm an der Stelle der komplexen Zahl dargestellt.

Die Grundwerte unseres Programms sehen folgendermaßen aus:

```
Realteil:               von -1.3 bis 2.64
Imaginärteil:           von -1.2 bis 1.2
maximale Iteration:     20
```

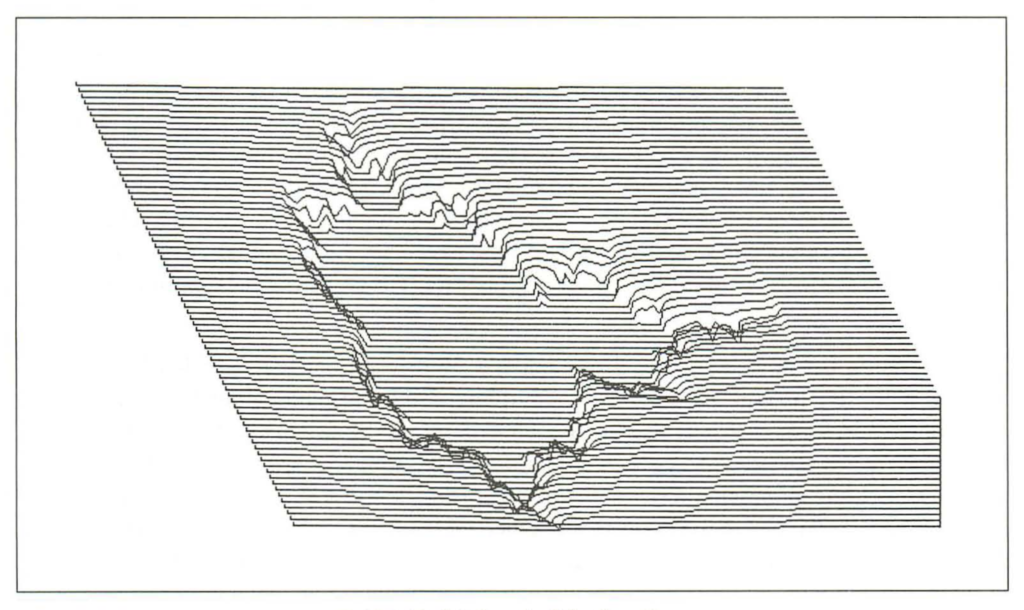


Bild 6.11: Linien-Apfelmännchen

Unser Beispielprogramm geht allerdings andere Wege: Auch hier wird die Berechnung nach der oben angegebenen Formel durchgeführt. Allerdings wird nicht eine Farbe am betreffenden Punkt dargestellt, sondern eine Liniengrafik (siehe Bild 6.11), die durch eine bestimmte Perspektive den Wert des Punktes erkennen läßt. Es handelt sich also um die in diesem Kapitel angesprochene Kavaliersperspektive. Sicher werden Sie so eine Darstellung der »Apfelmännchen« noch nicht gesehen haben.

Das Programm bietet aber noch mehr: Es wird nämlich nicht nur ein Bild erstellt, sondern es werden auch alle Koordinaten der Punkte gespeichert, aus denen sich dieses Bild zusammensetzt. Damit hat man die Möglichkeit, das Bild aus jedem beliebigen Blickwinkel darzustellen.

Noch eine Besonderheit des Programms: Im Berechnungsmodus haben Sie die Möglichkeit, das zur Zeit erstellte Bild in Schwarzweiß- oder in Rot-Grün-Grafik darzustellen. Wir haben Ihnen ja schon die Besonderheiten der Rot-Grün-Grafiken erläutert. Damit ist es nämlich möglich, auf einem zweidimensionalen Bildschirm dreidimensionale Bilder darzustellen. Da wir unser Apfelmännchen ja in drei Dimensionen berechnen (X-, Y- und Z-Koordinate), kann also eine solche Darstellungsweise genutzt werden. Allerdings müssen

wir zugeben, daß der 3-D-Effekt hier nicht so verblüffend ausfällt wie bei unseren speziellen Grafikdemos. Aber zum Ausprobieren reicht das allemal.

Die bisher erschienenen Berechnungsprogramme zur Mandelbrot-Menge sind im Laufe der Zeit immer leistungfähiger geworden. Überhaupt war dieses Anwendungsgebiet noch vor wenigen Jahren auf die Großrechner beschränkt. Mittlerweile sind die Programme für den Atari ST so leistungsfähig geworden, daß sie innerhalb weniger Sekunden ein vollständiges Bild berechnen. In unserem Programm wird dieses Bild als eine Liniengrafik berechnet. Auch unser Programm ist sehr schnell. Das liegt aber vor allem daran, daß nicht jeder Punkt berechnet wird, sondern nur jeder zweite, dritte, oder sogar nur jeder neunte Punkt auf der X- und der Y-Achse. Die Punkte werden dann durch Linien miteinander verbunden, wodurch das Objekt überhaupt erst sichtbar wird. Diese Methode ist natürlich nicht auf jeden Bereich der Mandelbrot-Menge anwendbar. So werden also manche Bereiche besser, andere dagegen schlechter dargestellt. Sollte Ihnen die Berechnung zu langsam vorkommen, lesen Sie doch bitte den Abschnitt über die Geschwindigkeitsoptimierungen im Kapitel 6.4.1.

Das Programm befindet sich unter dem Namen 6_3.GFA auf der beiliegenden Diskette. Es ist sowohl in der hohen, als auch in der mittleren Bildschirmauflösung lauffähig. In der mittleren Bildschirmauflösung haben Sie außerdem die Möglichkeit der Rot-Grün-Darstellung. Wie wir bereits dazu erklärten, benötigen Sie dann eine spezielle Rot-Grün-Brille. Diese Darstellungsmöglichkeit der 3-D-Grafiken ist schon seit längerem in Gebrauch. Wir gehen daher davon aus, daß Sie im Besitz einer solchen Rot-Grün-Brille sind oder aber in Ihrem Bekanntenkreis sich eine derartige Brille besorgen läßt. Möglicherweise kann Ihnen ja auch ein Optiker in Ihrer Stadt weiterhelfen. Nun muß aber auch fairerweise gesagt werden, daß die so ermöglichten 3-D-Effekte keinesfalls der Weisheit letzter Schluß sind. Oftmals ist überhaupt kein dreidimensionaler Effekt sichtbar. Manchmal müssen Sie erst durch Ausprobieren den richtigen Abstand zum Monitor feststellen. Eventuell muß die Helligkeit des Monitors verringert oder erhöht werden usw. Sie können auch im Programm in der Prozedur »Grundeinstellungen« die Farbwerte für Rot und Grün ändern – experimentieren Sie einfach ein wenig.

Bevor wir die Bedienung des Programms erklären, drucken wir jetzt das Listing des Programms 6_3.GFA ab. Einige Prozeduren haben wir auch hier wieder »eingeklappt«, da sie für einen vollständigen Abdruck einfach zu umfangreich sind und weniger wichtige Teile enthalten bzw. im Listing schon an anderer Stelle ausführlich erläutert wurden.

```
' *****                    PROGRAMM 6.3                    *****
' ****************************************************************
' *****                                                     *****
' *****            DREIDIMENSIONALE DARSTELLUNG             *****
' *****                DER MANDELBROT-MENGE                 *****
' *****                 (3D-APFELMÄNNCHEN)                  *****
' *****                                                     *****
' *****           WILHELM BESENTHAL UND JENS MUUS           *****
' *****                                                     *****
' *****            MARKT & TECHNIK, MÜNCHEN 1988            *****
' *****                                                     *****
' ****************************************************************
'
'
'
resource                          !Resource-Datei laden, Variablen belegen
grundeinstellungen                !wichtige Grundstellungen
REPEAT
  menu                            !Pull-Down-Menüs abfragen, Unterprog. aufrufen
  ON modus GOSUB info,laden,spei,lade3d,spei3d,ende,
                 werte,ausschni,rechne,zeige,rotation
UNTIL 0
END
'
'
```

Hier beginnen die Unterprogramme. Sie sehen, daß unser Hauptprogramm auch hier wieder sehr kurz ausgefallen ist. Im wesentlichen wird hier nur festgestellt, welche Funktionen der Anwender wünscht und dann in das entsprechende Unterprogramm verzweigt. Nach Ausführung der gewünschten Funktion wird wieder die Hauptprogrammschleife abgearbeitet.

Das erste Unterprogramm stellt eine Info-Dialogbox dar. Dazu wird noch eine zweite Prozedur »(zeig_dialogbox())« genutzt, die eine Dialogbox aus dem Resource mit der angegebenen Anfangsadresse auf dem Bildschirm zeichnet. Die Kontrolle über die Box wird durch »FORM_DO()« übernommen. Mit »OB_STATE()« wird der selektierte Button wieder normal dargestellt und anschließend die Dialogbox wieder gelöscht. Mit dem Befehl »PUT« wird dann der Bildschirminhalt wiederhergestellt.

Nach diesem Grundprinzip arbeiten alle unsere Routinen, die Dialogboxen darstellen. Nur wenn Daten in eine Box geschrieben oder ausgelesen werden müssen, gibt es noch weitere Programmschritte (die Info-Box hat lediglich einen »Weiter«-Button).

```
'
PROCEDURE info                          !Info-Dialogbox darstellen
  zeig_dialogbox(adresse2)
  ex%=FORM_DO(adresse2,0)
  OB_STATE(adresse2,weiter1&)=0
  loesch_dialogbox(adresse2)
  PUT 0,0,bild$
RETURN
```

Es folgen die Routinen, die das Einladen bzw. Speichern von Bilddateien ermöglichen.

```
PROCEDURE laden                         !Bilddatei von Disk laden
  fileselect(".APF",name$)
  IF name$="\" OR name$=""
  ELSE
    DEFMOUSE 2
    grafik_laden(name$,XBIOS(2),1)
    GET 0,0,639,399*ba,bild$
    reala=FLOAT{XBIOS(2)+31964}         !Werte aus Bildspeicher lesen
    reale=FLOAT{XBIOS(2)+31972}
    imaga=FLOAT{XBIOS(2)+31980}
    image=FLOAT{XBIOS(2)+31988}
    tmax%=CARD{XBIOS(2)+31996}
    IF tmax%>999                        !Maximale Tiefe begrenzen
      tm&=999
    ELSE
      tm&=tmax%
    ENDIF
    werte_in_dialogbox_werte            !Geladene Werte in Werte-Dialogbox
    DEFMOUSE 0
  ENDIF
RETURN
PROCEDURE spei                          !Bild auf Disk speichern
  fileselect(".APF",name$)
  IF name$>""
    DEFMOUSE 2
    PUT 0,0,bild$
    werte_aus_dialogbox_werte           !Werte aus Werte-Dialogbox holen
    FLOAT{XBIOS(2)+31964}=reala         !und in den Bildspeicher
    FLOAT{XBIOS(2)+31972}=reale
    FLOAT{XBIOS(2)+31980}=imaga
    FLOAT{XBIOS(2)+31988}=image
    CARD{XBIOS(2)+31996}=tm&
    BSAVE name$,XBIOS(2),32000          !Bild abspeichern
```

```
    DEFMOUSE 0
  ENDIF
RETURN
```

Zwei weitere Prozeduren dienen dazu, 3-D-Daten der Bilder zu laden oder zu speichern. Um das schnell zu bewerkstelligen, wird dabei ein kompletter Speicherbereich des ST auf die Diskette geschrieben. Bei diesem Bereich handelt es sich um das Variablen-Array, das die Daten enthält. Die Anfangsadresse erhalten wir durch »*pu&( )«.

```
PROCEDURE lade3d                        !Daten der 3-D-Punkte eines Bildes laden
  fileselect(".3D",name$)               !Dabei wird das Bild selbst nicht
  IF name$>""                           !eingeladen
    IF EXIST(name$)
      DEFMOUSE 2
      OPEN "I",#1,name$
      IF NOT EOF(#1)
        INPUT #1,pun&                   !Anzahl der Bildpunkte
        IF pun&>0
          adresse%={*pu&()}+8           !Anfangsadresse des Arrays
          laenge%=2*5*pun&              !Arraylänge berechnen
          BGET #1,adresse%,laenge%      !Block laden
        ENDIF
      ENDIF
      CLOSE #1
      DEFMOUSE 0
    ENDIF
  ENDIF
  bipunk&=pun&                          !Endpunkt für Rotation
RETURN
PROCEDURE spei3d                        !Daten der 3-D-Punkte eines
                                        !Bildes speichern
  fileselect(".3D",name$)               !Das Bild selbst jedoch nicht
                                        !speichern
  IF name$>""
    DEFMOUSE 2
    OPEN "O",#1,name$
    PRINT #1,pun&                       !Anzahl der Bildpunkte
    IF pun&>0
      adresse%={*pu&()}+8               !Anfangsadresse des Arrays
      laenge%=2*5*pun&                  !Berechnung der Arraylänge
      BPUT #1,adresse%,laenge%          !Block speichern
    ENDIF
    CLOSE #1
```

```
    DEFMOUSE 0
  ENDIF
RETURN
```

Die folgende Prozedur wird beim Verlassen des Programms aufgerufen. Sie stellt den Ursprungszustand des Speichers wieder her. Außerdem werden die zwei genutzten Farbregister wieder mit den Farben Weiß und Schwarz belegt.

```
PROCEDURE ende                         !Programm beenden
  ~MENU_BAR(adresse1,0)                !Menüleiste abschalten
  ~RSRC_FREE()                         !Resource-Speicher freigeben
  VSETCOLOR 0,7,7,7                    !Grundfarben wiederherstellen
  VSETCOLOR 1,0,0,0
  END
RETURN
```

Die Routine »werte« stellt eine Dialogbox dar (siehe Bild 6.12) und überwacht diese.

```
PROCEDURE werte                        !Werte-Dialogbox zeigen u. bearbeiten
  zeig_dialogbox(adresse3)             !Dialogbox darstellen
  REPEAT
    ex%=FORM_DO(adresse3,0)
    IF ex%=grundw&                     !Grundwerte einstellen
      grundwerte
      werte_in_dialogbox_werte
      OB_STATE(adresse3,grundw&)=0
      ~OBJC_DRAW(adresse3,0,8,x%,y%,w%,h%)
    ENDIF
  UNTIL ex%=weiter2&
  OB_STATE(adresse3,weiter2&)=0
  loesch_dialogbox(adresse3)           !Dialogbox löschen
  werte_aus_dialogbox_werte            !Aktuelle Werte auslesen
  PUT 0,0,bild$                        !Bild wiederherstellen
RETURN
PROCEDURE ausschni                     !Bildausschnitt mit Maus bestimmen
  WHILE MOUSEK>0
  WEND
  PUT 0,0,bild$
  DEFMOUSE 5
  REPEAT
    m_t&=MOUSEK
  UNTIL m_t&>0
  IF m_t&=1
    m_x1&=MOUSEX
    m_y1&=MOUSEY
```

```
    GRAPHMODE 3
    HIDEM
    WHILE MOUSEK=1
      m_x2&=MOUSEX                      !Koordinaten für das Rechteck
      m_z1&=MOUSEY                      !Bestimmen und berechnen
      m_b&=ABS(m_x2&-m_x1&)
      m_y2&=(m_y1&+m_b&*0.625*ba)
      m_z2&=m_y2&+ABS(m_z1&-m_y1&)
      m_x3&=m_x2&+(m_y2&-m_y1&)/2
      m_x4&=m_x3&-m_b&
      ausschnitt_zeigen                 !Rechteck zeichnen und
      ausschnitt_zeigen                 !wieder löschen
    WEND
    ausschnitt_zeigen
    SHOWM
    ALERT 2,"Ausschnitt übernehmen?",1,"JA|NEIN",var
    ausschnitt_zeigen
    IF var=1                            !Ausschnitt übernehmen, Werte berechnen
      dx=(reale-reala)/(639-2*xrand&(gross&))
      dy=(image-imaga)/((399*ba)-2*yrand&(gross&))
      tm&=ABS(m_z1&-m_y1&)
      reala=reala+(m_x1&-xrand&(gross&)-m_y1&/2+yrand&(gross&))*dx
      reale=reala+(m_x2&-m_x1&)*dx
      imaga=imaga+(m_y1&-yrand&(gross&))*dy
      image=imaga+0.625*ABS(reale-reala)
    ENDIF
    GRAPHMODE 1
  ENDIF
  PUT 0,0,bild$
  werte_in_dialogbox_werte              !Werte in die Dialogbox
  DEFMOUSE 0
RETURN
PROCEDURE rechne                        !Bild berechnen
  CLS                                   !Je nach Einstellung normale Darstellung
  HIDEM                                 !Oder Rot-Grün-Darstellung
  cx=reala                              !Für Rot-Grün-Brille
  cy=imaga
  dx=(reale-reala)/(639-2*xrand(gross&))
  dy=(image-imaga)/((399*ba)-2*yrand(gross&))
  CLR pun&,lin&
  zeit=TIMER
  FOR i&=yrand(gross&) TO (399*ba)-yrand(gross&) STEP genau&
    links&=pun&
```

```
offset&=i&/2-yrand(gross&) !Offset für schräge Darstellung
DRAW xrand(gross&)+offset&,i&
FOR j&=xrand(gross&) TO 639-xrand(gross&) STEP genau&
  CLR xw,yw,xq,yq
  FOR t&=1 TO tm&-1                   !Berechnungsroutine der Mandelbrotmenge
    yw=2*xw*yw-cy                     !(Für einen Punkt)
    xw=xq-yq-cx
    xq=xw*xw
    yq=yw*yw
    EXIT IF xq+yq>4
  NEXT t&
  IF t& >talt&                        !Koordinaten der Punkte
    IF talt& tm&                      !In die X-, Y- und Z-Arrays
      pu&(0,pun&)=j&
      pu&(1,pun&)=i&
      pu&(2,pun&)=t&
      DRAW  TO j&+offset&,i&+t&
      INC pun&
    ELSE
      pu&(0,pun&)=j&-1
      pu&(1,pun&)=i&
      pu&(2,pun&)=tm&
      INC pun&
      pu&(0,pun&)=j&
      pu&(1,pun&)=i&
      pu&(2,pun&)=t&
      DRAW  TO j&-1+offset&,i&+tm& TO j&+offset&,i&+t&
    ENDIF
  ENDIF
  talt&=t&
  cx=cx+dx*genau&
NEXT j&
CLR talt&
pu&(0,pun&)=j&-genau&
pu&(1,pun&)=i&
pu&(2,pun&)=t&
INC pun&
DRAW  TO 639-xrand(gross&)+offset&,i&+t&
rechts&=pun&
IF drei_d_flag&=TRUE                  !Bei Rot-Grün-Darstellung
  COLOR 2                             !Zusätzliches Bild in Grün
  FOR i%=links& TO rechts&-2
    rgo%=(pu&(1,i%)+pu&(2,i%)-i&)/5   !Räumlicher Effekt
```

```
        DRAW pu&(0,i%)+offset&+rgx&+rgo%,pu&(1,i%)+pu&(2,i%)+rgy&
                TOrgo%+pu&(0,i%+1)+offset&+rgx&,
                pu&(1,i%+1)+pu&(2,i%+1)+rgy&
      NEXT i%
      COLOR 1                               !Grundbild in Rot
    ENDIF
    cx=reala
    cy=cy+dy*genau&
    EXIT IF MOUSEK=2
  NEXT i&
  PRINT AT(2,2);TIMER-zeit
  GET 0,0,639,399*ba,bild$                  !Bildinhalt retten
  bipunk&=pun&                              !Endpunkt für 3-D-Rotation
  vipunk&=0
  SHOWM
RETURN
```

Das war die Berechnungsroutine für die Grafiken. Sie sehen, daß je nach gewünschtem Modus die Grafik normal oder in rot-grüner Farbe gezeichnet wird.

Die folgende Prozedur blendet lediglich die Menüzeile aus, um das komplette Bild zu zeigen. Wir haben diese Routine »eingeklappt«.

```
> PROCEDURE zeige                           !Ganzes Bild zeigen
```

Dafür ist die Rotationsprozedur hier abgedruckt. Es werden nach der Abarbeitung einer entsprechenden Dialogbox (Bild 6.13) alle Punkte einer Grafik neu berechnet, um einen neuen Blickwinkel zu erreichen. Die Grafik wird dann durch »draw_rotation« dargestellt und ein kleines Auswahlmenü gezeichnet. Durch Druck der rechten Maustaste wird wieder zum Hauptprogramm gesprungen.

```
PROCEDURE rotation                          !Rotation des 3-D-Körpers
  werte_in_dialogbox_rotation               !Vorher Dialogbox bearbeiten
  zeig_dialogbox(adresse4)
  ex%=FORM_DO(adresse4,0)
  OB_STATE(adresse4,zurueck&)=0
  OB_STATE(adresse4,weiter3&)=0
  loesch_dialogbox(adresse4)
  IF ex%=weiter3&
    FOR i%=vopunk& TO bipunk&               !Für Drehung um Mittelpunkt
      pu&(3,i%)=pu&(0,i%)-320
      pu&(4,i%)=pu&(1,i%)-y_offset&
      pu&(5,i%)=pu&(2,i%)
    NEXT i%
```

```
    werte_aus_dialogbox_rotation
    REPEAT
      CLS
      DEFMOUSE 2
      IF ma_mo&=0
        wx&=adsub&*dxw&
        c_px=COSQ(wx&)
        s_px=SINQ(wx&)
        FOR i&=vopunk& TO bipunk&        !X-Drehung
          p_a&=pu&(4,i&)
          pu&(4,i&)=c_px*pu&(4,i&)-s_px*pu&(5,i&)
          pu&(5,i&)=c_px*pu&(5,i&)+s_px*p_a&
          alt&=pu&(3,i&)
        NEXT i&
      ELSE IF ma_mo&=1
        wy&=adsub&*dyw&
        c_py=COSQ(wy&)
        s_py=SINQ(wy&)
        FOR i&=vopunk& TO bipunk&        !Y-Drehung
          p_a&=pu&(5,i&)
          pu&(5,i&)=c_py*pu&(5,i&)-s_py*pu&(3,i&)
          pu&(3,i&)=c_py*pu&(3,i&)+s_py*p_a&
        NEXT i&
      ELSE IF ma_mo&=2
        wz&=adsub&*dzw&
        c_pz=COSQ(wz&)
        s_pz=SINQ(wz&)
        FOR i&=vopunk& TO bipunk&        !Z-Drehung
          p_a&=pu&(3,i&)
          pu&(3,i&)=c_py*pu&(3,i&)-s_py*pu&(4,i&)
          pu&(4,i&)=c_py*pu&(4,i&)+s_py*p_a&
        NEXT i&
      ENDIF
      draw_rotation                      !Berechnete Grafik neu zeichnen
      maus_menu(ma_mo&)
    UNTIL MOUSEK>1 OR ma_mo&>2
  ELSE
    PUT 0,0,bild$
  ENDIF
  DEFMOUSE 0
RETURN
```

Es folgt eine kurze Routine, die eine Grafik von Diskette lädt. Die Routine erkennt dabei automatisch einige mögliche Dateiformate. Wie Sie sicher wissen, benutzen die verschiedenen Grafikprogramme meist auch verschiedene Formate, um die erstellten Bilder zu speichern. Zu den reinen Bilddaten kommen dabei meist noch die Farbdaten hinzu. Komprimierte Formate können mit dieser Routine allerdings nicht eingeladen werden.

```
PROCEDURE grafik_laden(bild_name$,bild_adresse,filenummer)
  IF EXIST(bild_name$)
    OPEN "I",#filenummer,bild_name$
    bild_laenge%=LOF(#filenummer)
    IF bild_laenge%>31999
      SELECT bild_laenge%
      CASE 32034                              !Degas-Format
        SEEK #filenummer,34
      CASE 32128                              !Neochrome-Format
        SEEK #filenummer,128
      ENDSELECT
      BGET #filenummer,bild_adresse,32000     !Alle Formate
    ELSE
      BLOAD bild_name$,bild_adresse           ! 32000 Bytes
    ENDIF
    CLOSE #filenummer
  ENDIF
RETURN
```

Die folgenden Prozeduren laden eine Resource-Datei von Diskette. Dabei ist ein bestimmter Pfadname voreingestellt. Sollte es einmal Schwierigkeiten dabei geben, die Resource-Datei einzuladen, wird das Programm nach einer Fehlermeldung abbrechen. Sie sollten dann den angesprochenen Pfadnamen (in der Prozedur »resource«) genau überprüfen. In unserem Programm nutzen wir die Funktion »DIR$(0)«, um den aktuellen Pfad zu ermitteln. Diesem Namen hängen wir noch den Ordner- und den Dateinamen unserer Programmdiskette an.

Anschließend werden Variablen belegt, die für die spätere Nutzung der GEM-Elemente (Objektbäume, Objekte) verwendet werden sollen. Zum Schluß ermitteln wir die Anfangsadressen unserer Objektbäume. Auch diese Routine ist hier nur eingeklappt angedeutet.

```
> PROCEDURE resource                    !Resource-File laden, Variablen belegen
> PROCEDURE resource_laden(rsc_pfad$)
```

Es folgt eine Prozedur, die zu Beginn des Programms aufgerufen wird und wichtige Grundvariablen belegt sowie andere grundsätzliche Programmschritte (z.B. Bildschirmauflösung feststellen usw.) enthält.

```
PROCEDURE grundeinstellungen            !Wichtige Anfangswerte
  aufl%=XBIOS(4)
  IF aufl%=0
    ALERT 3,"Bitte im Desktop|auf mittlere
             Auflösung|umschalten",1,"ENDE",var
    ende
  ENDIF
  IF aufl% 2
    ba=0.5
  ELSE
    ba=1
  ENDIF
  y_offset&=ba*200                      !Y-Offset für Zeichenroutine
  IF aufl%>2
    ende
  ENDIF
  IF aufl% 2                            !Je nach Auflösung 3-D-Feld selektierbar
    OB_STATE(adresse3,dreidfla&)=0
  ELSE
    OB_STATE(adresse3,dreidfla&)=8      !... oder nicht
  ENDIF
  rot%=6                                !Optimaler Farbwert für den Monitor
  gruen%=4                              !(muß eventuell geändert werden)
  VSETCOLOR 0,0,0,0                     !Hintergrund schwarz
  VSETCOLOR 1,7,7,7                     !Vordergrund weiß
  VSETCOLOR 2,7,7,7
  DIM xrand&(4),yrand&(4)               !Offset für verschiedene Bildgrößen
  xrand&(1)=0
  yrand&(1)=0
  xrand&(2)=80
  yrand&(2)=50*ba
  xrand&(3)=160
  yrand&(3)=100*ba
  xrand&(4)=280
  yrand&(4)=175*ba
  grundwerte                            !Apfelmännchengrundwerte
  gross&=3                              !Anfangswert Bildgröße
  werte_in_dialogbox_werte
  DIM pu&(5,20000)
  rgx&=5                                !Offset für Rot-Grün-Darstellung
  rgy&=1
  dxw&=30                               !Anfangswerte Winkel Rotation
  dyw&=30
```

```
  dzw&=30
  CLS
  DEFTEXT 1,0,0,13*ba
  TEXT 20,20*ba,"  X >"                  !Kleines Menübild (Rotation) zeichnen
  TEXT 20,37*ba,"  Y >"
  TEXT 20,55*ba,"  Z >"
  FOR i&=0 TO 2
    BOX 15,(6+i&*1)*ba,31,(23+i&*17)*ba
    BOX 47,(6+i&*17)*ba,63,(23+i&*17)*ba
  NEXT i&
  GET 15,6*ba,63,57*ba,m_menu$
  CLS
  GET 0,0,639,399*ba,bild$
RETURN
PROCEDURE menu                           !Pull-down-Menü abfragen
  ~MENU_BAR(adressel,1)
  ~EVNT_MESAG(mesag_addesse)
  ~MENU_TNORMAL(adressel,MENU(4),1)
  IF MENU(4)=desk&
    IF MENU(5)=info&
      modus=1
    ENDIF
  ENDIF
  SELECT MENU(4)
  CASE datei&
    SELECT MENU(5)
    CASE laden&
      modus=2
    CASE spei&
      modus=3
    CASE lade3d&
      modus=4
    CASE spei3d&
      modus=5
    CASE ende&
      modus=6
    ENDSELECT
  CASE bilre&
    SELECT MENU(5)
    CASE werte&
      modus=7
    CASE ausschni&
      modus=8
```

```
    CASE rechne&
      modus=9
    ENDSELECT
  CASE bilan&
    SELECT MENU(5)
    CASE zeige&
      modus=10
    CASE rotation&
      modus=11
    ENDSELECT
  ENDSELECT
RETURN
PROCEDURE zeig_dialogbox(adressnr)
  ~FORM_CENTER(adressnr,x%,y%,w%,h%)
  ~FORM_DIAL(0,0,0,0,0,x%,y%,w%,h%)
  ~FORM_DIAL(1,0,0,0,0,x%,y%,w%,h%)
  ~OBJC_DRAW(adressnr,0,8,x%,y%,w%,h%)
RETURN
PROCEDURE loesch_dialogbox(adressnr)
  ~FORM_CENTER(adressnr,x%,y%,w%,h%)
  ~FORM_DIAL(2,0,0,0,0,x%,y%,w%,h%)
  ~FORM_DIAL(3,0,0,0,0,x%,y%,w%,h%)
RETURN
> PROCEDURE grundwerte                  !Werte für das Grundbild
> PROCEDURE fileselect(extension$,VAR file$)
PROCEDURE werte_in_dialogbox_werte
  IF drei_d_flag&=TRUE
    OB_FLAGS(adresse3,dreidfla&)=1
  ENDIF
  CHAR{{OB_SPEC(adresse3,real1&)}}=STR$(reala)
  CHAR{{OB_SPEC(adresse3,real2&)}}=STR$(reale)
  CHAR{{OB_SPEC(adresse3,imag1&)}}=STR$(imaga)
  CHAR{{OB_SPEC(adresse3,imag2&)}}=STR$(image)
  CHAR{{OB_SPEC(adresse3,tiefe&)}}=STR$(tm&)
  CHAR{{OB_SPEC(adresse3,aufloes&)}}=STR$(genau&)
RETURN
PROCEDURE werte_aus_dialogbox_werte
  reala=VAL(CHAR{{OB_SPEC(adresse3,real1&)}})
  reale=VAL(CHAR{{OB_SPEC(adresse3,real2&)}})
  imaga=VAL(CHAR{{OB_SPEC(adresse3,imag1&)}})
  image=VAL(CHAR{{OB_SPEC(adresse3,imag2&)}})
  tm&=VAL(CHAR{{OB_SPEC(adresse3,tiefe&)}})
  genau&=VAL(CHAR{{OB_SPEC(adresse3,aufloes&)}})
```

```
  gross&=1
  REPEAT
    EXIT IF BTST(OB_STATE(adresse3,gr1&),0)
    INC gross&
    EXIT IF BTST(OB_STATE(adresse3,gr2&),0)
    INC gross&
    EXIT IF BTST(OB_STATE(adresse3,gr4&),0)
    INC gross&
    EXIT IF BTST(OB_STATE(adresse3,gr8&),0)
  UNTIL 0
  IF BTST(OB_STATE(adresse3,dreidfla&),0)=TRUE
    drei_d_flag&=TRUE                  !Je nach Auflösung rot-grün ...
    VSETCOLOR 1,rot%,0,0
    VSETCOLOR 2,0,gruen%,0
  ELSE
    drei_d_flag&=FALSE                 !oder schwarzweiß
    VSETCOLOR 1,7,7,7
    VSETCOLOR 2,7,7,7
  ENDIF
RETURN
> PROCEDURE ausschnitt_zeigen          !Rechteck für Ausschnitt zeichnen
PROCEDURE werte_in_dialogbox_rotation
  CHAR{{OB_SPEC(adresse4,xwi&)}}=STR$(dxw&)
  CHAR{{OB_SPEC(adresse4,ywi&)}}=STR$(dyw&)
  CHAR{{OB_SPEC(adresse4,zwi&)}}=STR$(dzw&)
  CHAR{{OB_SPEC(adresse4,vopun&)}}=STR$(vopunk&)
  CHAR{{OB_SPEC(adresse4,bipun&)}}=STR$(bipunk&)
  CHAR{{OB_SPEC(adresse4,maxpun&)}}=STR$(pun&)
RETURN
PROCEDURE werte_aus_dialogbox_rotation
  dxw&=VAL(CHAR{{OB_SPEC(adresse4,xwi&)}})
  dyw&=VAL(CHAR{{OB_SPEC(adresse4,ywi&)}})
  dzw&=VAL(CHAR{{OB_SPEC(adresse4,zwi&)}})
  vopunk&=MIN(VAL(CHAR{{OB_SPEC(adresse4,vopun&)}}),pun&)
  bipunk&=MIN(VAL(CHAR{{OB_SPEC(adresse4,bipun&)}}),pun&)
RETURN
PROCEDURE maus_menu(VAR m_modus&)
  PUT 570,330*ba,m_menu$               !Bei Rotation kleines Menü zeichnen ...
  DEFMOUSE 0
  REPEAT
    REPEAT                             !und abfragen
      m_t&=MOUSEK
    UNTIL m_t&>0
```

```
    EXIT IF m_t&=2
    m_x&=MOUSEX
    m_y&=MOUSEY
  UNTIL m_x&>570 AND m_x& 618 AND m_y&>330*ba AND m_y& 380*ba
  m_modus&=(m_y&-(330*ba))/17
  IF m_x&>594
    adsub&=1
  ELSE
    adsub&=-1
  ENDIF
  IF m_t&=2
    ma_mo&=3
  ENDIF
RETURN
```

Damit ist das Listing unseres Apfelmännchenprogramms beendet. Sie haben sicher Verständnis dafür, daß wir einige Routinen nicht vollständig angedruckt haben – ein Buch voller Listings war nicht unser Ziel. Die Programme befinden sich ja auf der beiliegenden Diskette.

Kommen wir zur Bedienung unseres Programms. Anders als bei den herkömmlichen Berechnungsprogrammen für die Mandelbrot-Menge werden in unserem Programm keine verschiedenen Farben für die Darstellung der Punkttiefe genutzt, sondern die Abweichung einer Linie in die Y-Richtung. Der Vorteil dieser Methode liegt darin, daß hierbei wesentlich weniger Punkte berechnet werden müssen, um ein leicht erkennbares Bild zu erzeugen.

Nach dem Programmstart erscheint eine Menüleiste mit den Einträgen:

Desk Datei Bild berechnen Bild anzeigen

Wundern Sie sich bitte nicht über unsere Farbauswahl, schwarzer Hintergrund und weiße Schrift. Für die Darstellung der Figuren ist diese Kombination einfach besser geeignet. Außerdem kann ja mit dem Programm auch eine Rot-Grün-Darstellung gemacht werden, die unbedingt einen schwarzen Hintergrund benötigt. Nach dem Programmende wird wieder auf normale Farben umgeschaltet. Kommen wir zu den Einträgen in der Menüleiste und den Pull-down-Menüs.

1. Pull-down-Menü »Desk«

Hier finden Sie den Eintrag INFO. Klicken Sie dieses Feld an, so zeigt sich Ihnen eine Informationsbox. Durch den Weiter-Button wird die Box wieder gelöscht.

2. Pull-down-Menü »Datei«

Der erste Eintrag in diesem Pull-down-Menü (Bild laden) dient dazu, ein auf der Diskette befindliches Bild zu laden. Dabei sollte es sich natürlich um ein zuvor erstelltes Apfelmännchenbild handeln. Allerdings können hierdurch auch fast alle anderen beliebigen

Bilddateien eingeladen werden. Leider können keine komprimierten Dateien geladen werden. Wenn es sich bei dem Bild um ein zuvor durch dieses Programm berechnetes Bild handelt, befinden sich innerhalb der Bilddaten zusätzlich noch die Berechnungswerte für unsere Berechnungsroutine. Dadurch können Sie später immer wieder feststellen, um welchen Bereich der komplexen Zahlen es sich bei dem vorliegenden Bild handelt. Es können dann auch weitere Berechnungen mit diesem Bild durchgeführt werden. Unsere Dateien haben grundsätzlich einen Umfang von 32.000 Byte. Für die Werte nutzen wir die letzten Bytes dieser Bilddaten. Dadurch geht natürlich ein kleiner Bereich des Bildes verloren (erkennbar am Punktmuster am unteren Bildrand), was sich aber kaum nachteilig bemerkbar macht.

Der folgende Menüeintrag heißt »Bild speichern« und dient dazu, die Bilddaten sowie die Berechnungswerte auf Diskette zu schreiben. In diesen Routinen erscheint jeweils eine Fileselect-Box. Als Extension ist ».BIL« voreingestellt, was Sie jedoch beliebig ändern können.

Zwei weitere Menüeinträge dienen dazu, nicht das Bild, sondern die Koordinatenwerte aller Bildpunkte zu speichern bzw. zu laden. Die Einträge heißen 3-D-Punktdaten laden »und 3-D-Punktdaten speichern«. Wie schon angesprochen, wird jede Koordinate (X-, Y- und Z-Koordinate) eines jeden Punktes im Speicher festgehalten, um im weiteren Verlauf des Programms die Berechnung von verschiedenen Betrachtungswinkeln zu ermöglichen. Die Daten stehen dazu in einem Variablenfeld, das komplett als Speicherblock auf Diskette gespeichert bzw. von Diskette eingelesen werden kann. Dabei bleibt der Bildinhalt natürlich unverändert. Sollen also alle Daten eines Bildes gespeichert werden, müssen Sie zwei Dateien (1. Bild, 2. Koordinaten) bilden.

Der letzte Eintrag im Pull-down-Menü »Datei« heißt »Programmende« und dient dazu, das Programm ordnungsgemäß zu beenden. Es werden dann zum Beispiel Speicherreservierungen rückgängig gemacht und die Farbregister mit den üblichen Werten belegt.

3. Pull-down-Menü »Bild berechnen«

Hier befinden sich drei Einträge. Der erste, »Werte ändern«, zeichnet eine Dialogbox (Bild 6.12), in der die Berechnungsdaten eines Bildes stehen und geändert werden können. Hier können Sie also die Werte für den Realwert, den Imaginärwert eines Bereiches der komplexen Zahlen sowie die Berechnungstiefe angeben. Die weiteren Einträge in der Dialogbox dienen dazu, die Auflösung festzulegen. Dabei handelt es sich um einen Wert von 1 bis 9, wobei 4 der Normalwert ist. Je größer dieser Wert ist, desto ungenauer ist die Berechnung, die dann aber auch erheblich schneller wird. Es handelt sich bei dem Auflösungswert einfach um den Stepwert der Berechnungsroutinen. Sie können in der Dialogbox auch noch die Bildgröße angeben ($^1/_1$, $^1/_2$, $^1/_4$ und $^1/_8$ Bildgröße), die ebenfalls einen wesentlichen Einfluß auf die Berechnungsgeschwindigkeit hat. Die kleinen Bilder können z.B. für einen schnellen »Berechnungstest« genutzt werden.

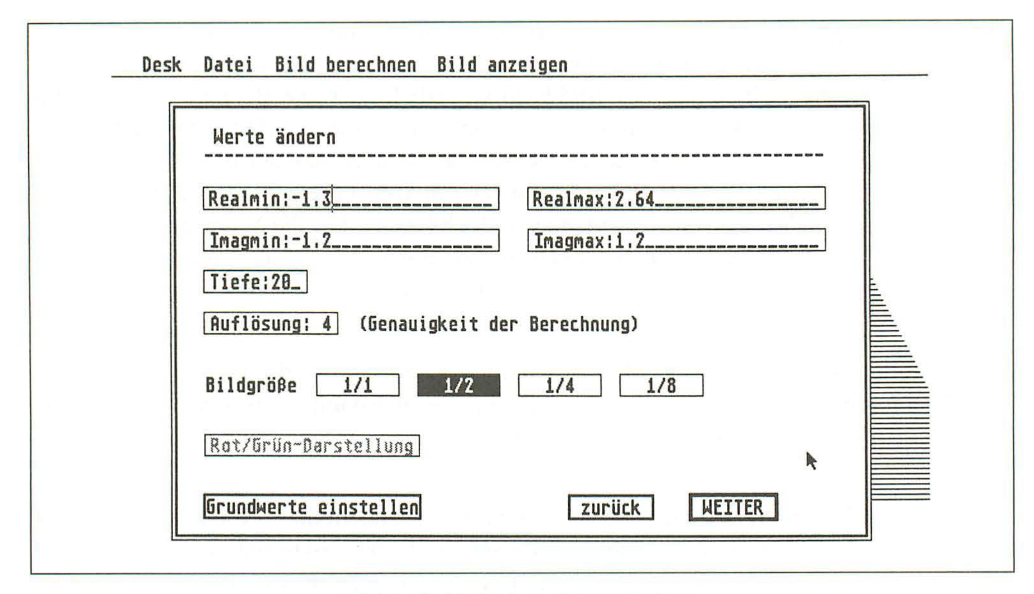


Bild 6.12: Dialogbox »Werte ändern«

Wenn Sie dieses Programm mit einem Farbmonitor betreiben, ist in der Dialogbox »Werte ändern« noch ein weiterer Button selektierbar: Rot-Grün-Darstellung. Sie können dadurch die in diesem Kapitel schon angesprochene Rot-Grün-Darstellung einer dreidimensionalen Grafik – hier die Apfelmännchen – einschalten. Sie bemerken das daran, daß von nun an alle Texte auf dem Bildschirm rot dargestellt werden. Wird dann ein Bild berechnet, erscheinen auch die grünen Linien. So ein Rot-Grün-Bild kann dann auch abgespeichert werden, allerdings kann keine Rot-Grün-Rotation (im Pull-down-Menü »Bild anzeigen«) durchgeführt werden. Um so ein Bild zu betrachten, brauchen Sie die schon angesprochene Brille. Bitte erwarten Sie von dieser Darstellungsart nicht zu viel. Oftmals stellt sich kaum ein bemerkenswerter 3-D-Effekt ein. Eine Besserung kann sich auch ergeben, wenn die Farbhelligkeit bei der Erstellung der Grafik geändert wird. Diese Einstellung können Sie in der Prozedur »grundeinstellungen« vornehmen. Vielleicht haben Sie ja schon die optimalen Werte in unserem Programm 6_1.GFA festgestellt und können sie hier übernehmen. Wenn Sie nur den Schwarzweiß-Monitor verwenden, kann die Rot-Grün-Darstellung natürlich nicht eingeschaltet werden.

Durch Anklicken des entsprechenden Feldes können Sie auch die Grundeinstellungen (die Startwerte) wiederherstellen.

Der zweite Eintrag im Pull-down-Menü »Bild berechnen«, »Ausschnitt bilden«, kann genutzt werden, um aus einem Bild mit der Maus einen Ausschnitt zu definieren, der dann in einer weiteren Berechnung neu berechnet wird. Es werden dann auch die neuen Werte in der Dialogbox »Werte ändern« eingetragen.

Der letzte und wichtigste Eintrag heißt »Bild berechnen«. Damit starten Sie die Berechnungsroutine. Gleichzeitig werden die berechneten Linien auf dem Bildschirm sichtbar. Es werden dabei nur jeweils fertige Linien gezeichnet, also nicht Punkt für Punkt. Somit kann es auch vorkommen, daß zeitweilig auf dem Bildschirm nichts geschieht. Aber keine Angst, das Programm ist dann natürlich immer noch bei der Berechnung. Wenn in der Dialogbox »Werte ändern« das Feld Rot-Grün-Darstellung angeklickt wurde, wird jetzt jede berechnete Linie durch zwei Farben (Rot und Grün) dargestellt. Dadurch kann sich, mit einer Rot-Grün-Brille betrachtet, der dreidimensionale Effekt bilden.

4. Pull-down-Menü »Bild anzeigen«

Wird in diesem Pull-down-Menü der erste Eintrag (ganzes Bild zeigen) angeklickt, wird die Menüzeile bis zum nächsten Maustastendruck ausgeblendet.

Das Anklicken des Feldes »Rotation« läßt die folgende Dialogbox auf dem Bildschirm erscheinen (Bild 6.13).

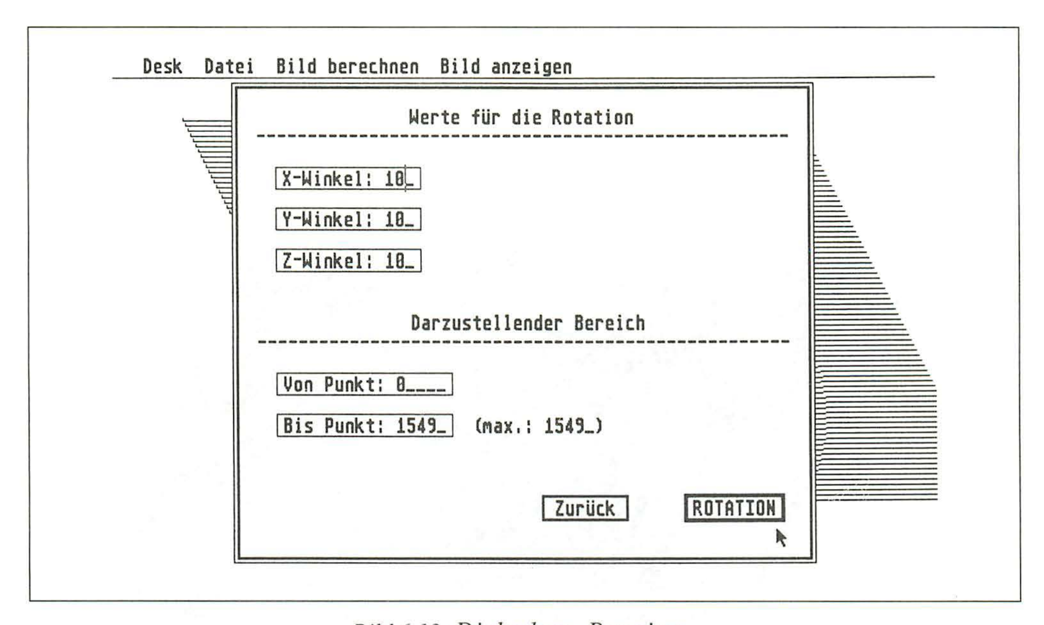


Bild 6.13: Dialogbox »Rotation«

Wie schon angesprochen, kann das berechnete Bild, oder auch nur Teile davon, aus beliebigen Blickwinkeln dargestellt werden. Dazu dient die folgende Dialogbox. Allerdings kann hier jeweils nur ein Winkelwert angegeben werden, um den sich die neue Darstellung von der vorherigen unterscheidet. Als Grundwerte sind jeweils 30 Grad angegeben. Im unteren Bereich der Box können Sie angeben, welcher Bereich der Punkte (Eckpunkte der Linien) dargestellt werden soll. Wenn nun der Button »Rotation« angeklickt wird, wird das erste Bild berechnet, was eventuell einen Moment dauert (der Mauscursor wird als

»Fliege« dargestellt). Wenn die Berechnung beendet ist, erscheint das Bild und zusätzlich in der unteren rechten Ecke ein kleines Menü, das mit der Maus bedient werden kann. Hier können Sie den Darstellungswinkel der drei Achsen (Drehung jeweils um den Mittelpunkt) verringern oder erhöhen. Durch einen Druck der rechten Maustaste wird die Rotationsroutine wieder verlassen.

Soviel zur Bedienung des Apfelmännchenprogramms. In Bild 6.14 sehen Sie ein weiteres Beispielbild, an dem das unkompilierte Programm übrigens (1/4 Bildgröße, Auflösung 4) etwa 9 Minuten rechnete. Dieses Bild ist auch auf der beiliegenden Diskette abgespeichert und trägt den Namen BEISPIEL.APF. Außerdem befindet sich auf der Diskette die 3-D-Datei zu diesem Bild (BEISPIEL.3-D).

Probieren Sie auch einmal folgenden Zahlenbereich:

```
Realmin:  0.546406       Realmax:  0.562461
Imagmin:  0.545057       Imagmax:  0.555091
Tiefe:  50
```

Wir wünschen Ihnen viel Spaß bei Ihrer Entdeckungstour durch das Apfelmännchen. Die Ausschnittfunktion des Programms wird Ihnen sicher ständig neue interessante Bereiche des Bildes zeigen.

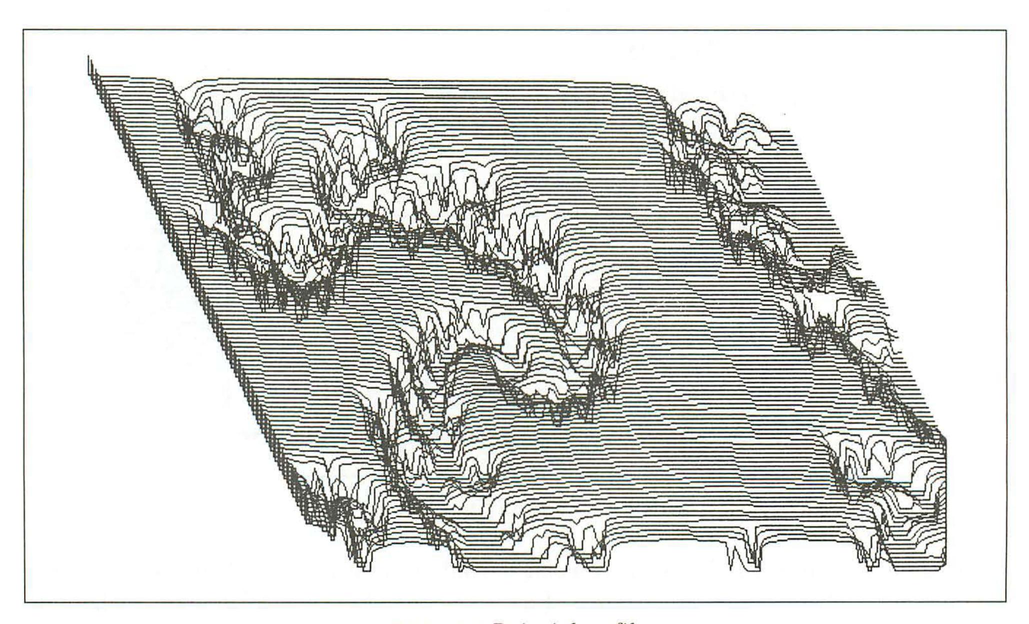


Bild 6.14: Beispielgrafik

6.4.3 Aussichten

Die zwei vorstehenden Grafikprogramme »Raytrace« und »Apfelmännchen« nehmen viel Raum in diesem Buch ein. Trotzdem könnte man noch sehr viel mehr zu diesen Themen bringen. Das geht so weit, daß wir uns auch hier wieder sagen müssen, was so oft in den Büchern steht: Weitere Ausführungen würden den Rahmen dieses Buches sprengen. Das ist natürlich immer eine bedauerliche Situation. Allerdings glauben wir, Ihnen das nötige Rüstzeug mit auf den Weg gegeben zu haben, um eigene Ideen in Programmen zu verwirklichen. Unser Buch kann und soll kein Basic-Kurs sein, vielmehr möchten wir Sie dazu ermutigen, kreativ ans Werk zu gehen. Auch hier ist nichts unmöglich, die notwendigen Informationen erhalten Sie in diesem Buch und natürlich in der Anleitung zum GFA-Basic 3.0. Mit dieser Programmiersprache kann sowohl der Anfänger als auch der fortgeschrittene Programmierer leistungsstarke Routinen schreiben.

Wie wäre es zum Beispiel mit Folgendem: Das Raytrace-Programm (6_2.GFA) errechnet nicht nur ein Bild, sondern 10 oder 20 Bilder, jedes mit denselben Seitenwänden. Bei jedem Bild werden jedoch die Koordinaten der reflektierenden Kugel(n) um einen geringen Wert geändert. Die daraus resultierenden Bilder lädt man nun in den Rechnerspeicher und stellt sie nacheinander auf dem Bildschirm dar. Ähnlich könnte man mit dem Apfelmännchenprogramm (6_3.GFA) mehrere Bilder errechnen und nacheinander auf dem Bildschirm (evtl. als Endlosschleife) zeigen. Man könnte auf diese Weise einen richtigen kleinen Film erzeugen, der einen kleinen Einblick in die bizarre Computerwelt darstellt. Das wäre doch interessant, oder?

6.5 Objektorientiertes Zeichenprogramm

Die diversen Malprogramme für den Atari ST kennen Sie sicher zur Genüge. Mit diesen Programmen können, je nach Bedienungskomfort, mehr oder weniger komfortable Bilder oder Geschäftsgrafiken erstellt werden. Nun arbeiten diese Programme in der Regel *pixelorientiert*. Das heißt, Punkt für Punkt wird auf dem Bildschirmhintergrund in einer beliebigen Farbe gesetzt. Diese Punkte sind dann sozusagen unverrückbar. Die meisten Grafikprogramme bieten für die anschließende Bearbeitung des erstellten oder digitalisierten Bildes natürlich diverse Funktionen an. So sind die einzelnen Punkte also doch mehr oder weniger umständlich editierbar. Der Vorteil der Grafikprogramme liegt darin, ganze Bilder oder Ausschnitte davon zu beeinflussen (z.B. zu verzerren).

Doch was ist, wenn Sie z.B. in einem Bild einige Kreise oder andere Objekte löschen möchten, oder ein Objekt ohne den Hintergrund zu beeinflussen, verschieben möchten. In der Regel sind Sie dann mit einem herkömmlichen pixelorientierten Grafikprogramm ziemlich aufgeschmissen. Es können zwar rechteckige Bildausschnitte verschoben werden, bei einem Kreis wird die Sache aber schon schwierig. Lupen- und Radiergummi-Funktion erleichtern zwar die Arbeit, schön wäre es jedoch, wenn jedes komplette Grafikobjekt einzeln angesprochen und verändert werden könnte. Nun ist der Begriff Objekt ja schon einige

Male gefallen, und dieser Begriff enthält auch schon die Lösung unseres Problems: Das Grafikprogramm sollte *objektorientiert* sein. Dann kann nämlich jedes Grafikobjekt (Punkt, Linie, Kreis usw.) einzeln und jederzeit beliebig bearbeitet, also geändert werden. Das objektorientierte Zeichenprogramm speichert die Bilddaten nicht im Bildschirmspeicher ab, sondern in einem speziellen Objektspeicher. Erst wenn die Objekte auf dem Bildschirm erscheinen sollen, werden sie vom Programm komplett neu gezeichnet.

Auch solche objektbezogenen Zeichenprogramme gibt es natürlich schon (z.B. GEM Draw oder Easy-Draw). Wir wollen Ihnen nun zeigen, wie ein derartiges Programm in GFA-Basic realisiert werden kann. Dazu liefern wir das Programm natürlich auch gleich mit. Es ist auf der beiliegenden Diskette unter dem Namen 6_4.GFA gespeichert. Am besten laden Sie das objektbezogene Zeichenprogramm jetzt einmal ein.

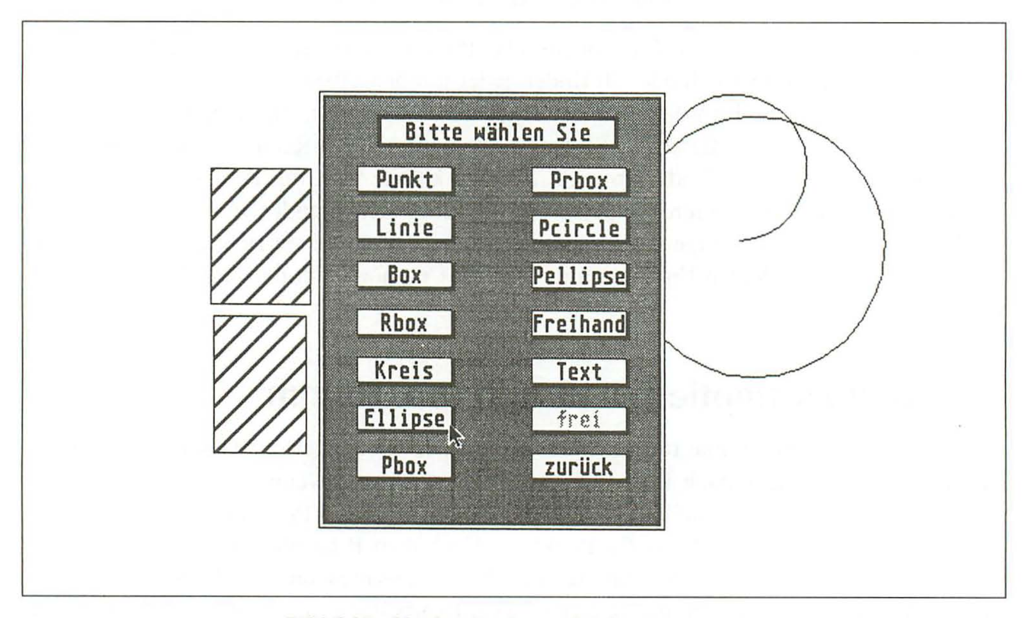


Bild 6.15: Objektorientiertes Zeichenprogramm

Nach dem Programmstart erscheint eine Menüleiste mit den Einträgen »Desk«, »Datei«, »Objekt« und »Parameter«. Unter dem Menütitel »Objekt« klicken Sie jetzt bitte im Pull-down-Menü den Eintrag »neue Objekte« an. Es erscheint dann die Auswahlbox, wie in Bild 6.15 zu sehen ist. Sie können dort durch Anklicken des gewünschten Feldes die einzelnen Funktionen aufrufen. Danach verschwindet die Auswahlbox vom Bildschirm, der Mauscursor wird zum Fadenkreuz, und Sie haben die Möglichkeit, ein oder mehrere Objekte mit der Maus zu definieren. Um z.B. ein Rechteck zu zeichnen, positionieren Sie den Mauscursor in die linke obere Ecke des Rechtecks. Dann drücken Sie die linke Maustaste und halten sie gedrückt. Jetzt kann durch Verschieben der Maus die gewünschte Größe des

Rechtecks eingestellt werden. Wird die linke Taste dann losgelassen, ist das Rechteck fertig definiert, und es kann das nächste Rechteck gezeichnet werden. Durch einen Druck der rechten Maustaste kommen Sie zurück zum Ausgangsmenü.

Es ist also hier die gleiche Vorgehensweise wie in den meisten anderen Zeichenprogrammen angewandt, um Grundfiguren zu erstellen. Allerdings gibt es eine Ausnahme, und das sind die Kreisfunktionen von »CIRCLE« und »ELLIPSE«. Dazu muß erwähnt werden, daß hierbei jeweils auch Kreisausschnitte quasi in einem »Arbeitsgang« definiert werden können. Sie müssen, um z.B. einen Kreis zu zeichnen, zunächst den Mittelpunkt festlegen. Dann drücken Sie die linke Maustaste und halten sie fest. Jetzt kann der Radius des Kreises durch das seitliche Bewegen (bei der Ellipse beide Radien durch Bewegen der Maus) eingestellt werden. Wenn Sie nun die linke Taste loslassen, erscheint Ihr definierter Kreis in seiner vollen Pracht, allerdings blinkend. Wenn Sie genau hinschauen, blinkt dabei nur ein Teil des Kreisumfangs, und diesen Teil können Sie durch Bewegen der Maus beliebig einstellen. Es handelt sich dabei um den Teil, der als Kreisausschnitt später stehenbleibt. Ist die gewünschte Größe eingestellt, drücken Sie nur noch einmal die linke Maustaste (evtl. einen Moment festhalten), und das Objekt ist fertig. Diese Vorgehensweise ist, zugegeben, etwas gewöhnungsbedürftig. Beim »Blinken« des Kreisausschnitts bleiben übrigens immer einige Punkte stehen. Das ist ein unbeabsichtigter, aber recht angenehmer Nebeneffekt, der sich ergibt, wenn auf einem Kreis ein Kreisausschnitt mit gleichem Radius im Grafikmodus 3 gezeichnet wird.

Die Stärke des objektorientierten Zeichenprogramms liegt in der Änderbarkeit der bestehenden Objekte. Ihnen steht dazu der Eintrag »Objekt bearbeiten« zur Verfügung. Wird dieses Feld im Pull-down-Menü »Objekte« angeklickt, erscheint die in Bild 6.16 gezeigte Dialogbox.

Um ein bestimmtes Objekt zu verändern, muß es zunächst angewählt und hervorgehoben werden. Das können Sie entweder dadurch machen, daß Sie in der Dialogbox eine Nummer angeben und das dazugehörige Feld »zeige Objekt Nummer« anklicken, oder dadurch, daß Sie alle Objekte der Reihe nach anwählen (Eintrag »Objekt auswählen« anklicken). Die Objekte sind fortlaufend durchnumeriert, wobei die jeweilige Nummer nicht untrennbar mit dem Objekt verbunden ist. So kann es also vorkommen, daß die Objekte neu durchnumeriert werden, wenn z.B. eines gelöscht wird. Es werden also nun alle Objekte der Reihe nach blinkend gezeichnet. Die jeweilige Objektnummer wird dabei in der linken oberen Ecke angezeigt. »Weiterschalten« können Sie durch Druck der linken Maustaste. Wenn Sie das gewünschte Objekt gefunden haben, drücken Sie bitte die rechte Maustaste (evtl. einen Moment festhalten). Es erscheint dann die Objekt-Informationsbox (Bild 6.17).

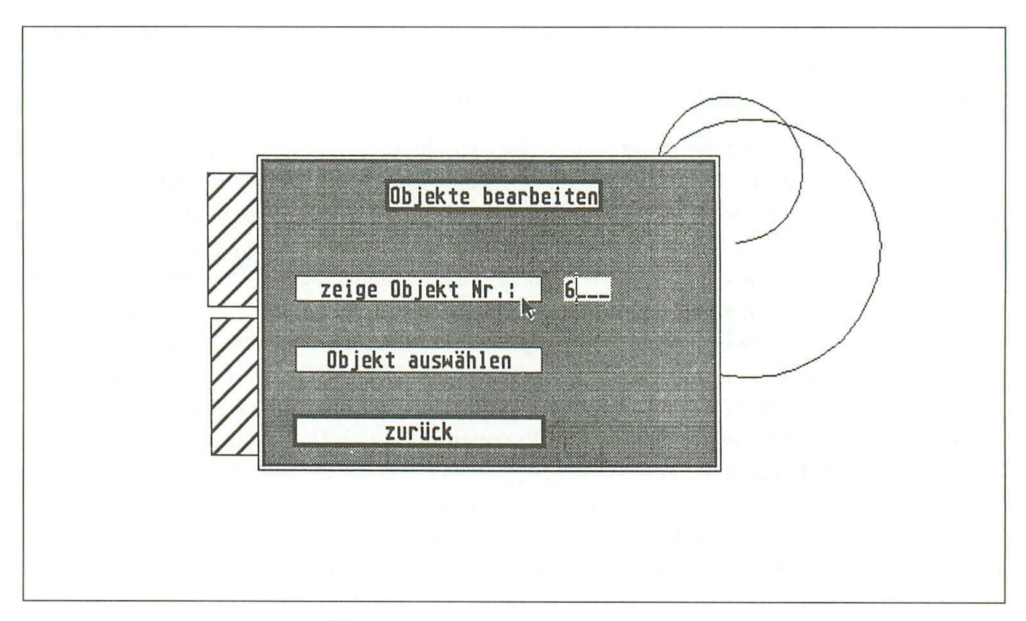


Bild 6.16: Dialogbox »Objekte bearbeiten«

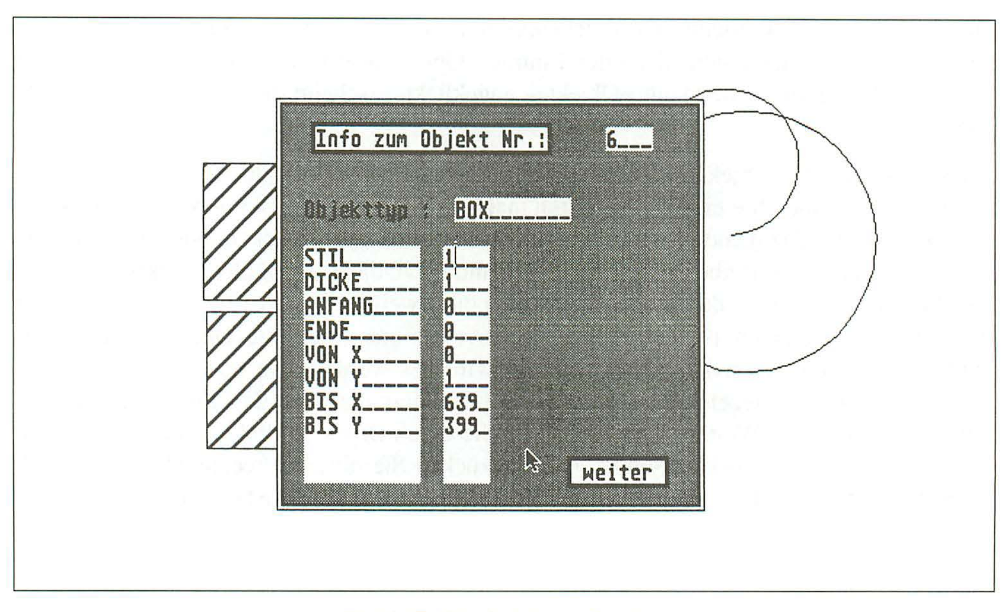


Bild 6.17: Objekt-Informationsbox

In dieser Dialogbox sind alle Informationen zu einem Objekt enthalten. Dabei können alle Werte, die das Aussehen des Objektes bestimmen, geändert werden. So kann zum Beispiel die Position geändert werden oder die Farbe oder andere Dinge.

Zum Thema Farbe muß gleich angemerkt werden, daß dieses Programm für den Schwarzweiß-Betrieb geschrieben wurde. Natürlich kann es auch in der mittleren Auflösung betrieben werden, es sind dabei jedoch Einschränkungen unumgänglich.

Im Pull-down-Menü »Objekte« finden Sie den Eintrag »lösche Objekt«. Wie der Name schon sagt, kann man hier entweder ein Objekt oder sogar alle Objekte löschen. Die Nummer des einzelnen Objektes ist natürlich auch hier wieder beliebig einstellbar.

Ein weiteres Feld im Pull-down-Menü »Objekte« heißt »Objektinfo« und zeichnet nach dem Anklicken eine Dialogbox, die Informationen über die Gesamtzahl der Objekte sowie den dadurch verbrauchten Speicherplatz enthält.

Im letzten Pull-down-Menü »Parameter« haben Sie die Möglichkeit, die derzeit eingestellten Linien-, Füll- und Farbparameter einzusehen und zu ändern. Dazu werden entsprechende Dialogboxen gezeichnet (siehe auch Bild 6.18).

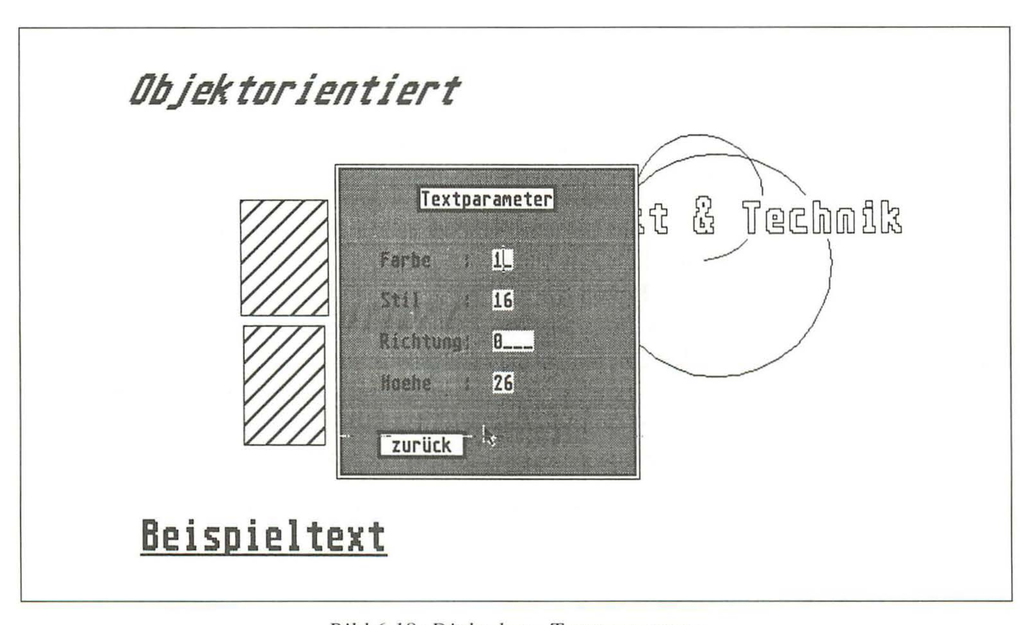


Bild 6.18: Dialogbox »Textparameter«

Wir drucken jetzt das Listing des Programms 6_4.GFA (objektorientiertes Zeichenprogramm). Wir haben auch hier wie schon bei den vorherigen Listings auf den kompletten Abdruck verzichtet. So sind also einige Prozeduren »eingeklappt« dargestellt.

```
' ************************ PROGRAMM 6.4 ************************
' **************************************************************
' *****                                                    *****
' *****          OBJEKTORIENTIERTES  ZEICHENPROGRAMM       *****
' *****                                                    *****
' *****            WILHELM BESENTHAL UND JENS MUUS         *****
' *****                                                    *****
' *****              MARKT & TECHNIK, MÜNCHEN 1988         *****
' *****                                                    *****
' **************************************************************
'
'
'
resource_laden(DIR$(0)+"\6.GFA\6_4.RSC")
resource_variablen
vorbereitungen
menu !Hauptprogrammschleife
'
```

Diese vier Programmteile machen schon das ganze Programm aus. Dabei handelt es sich natürlich um Prozeduraufrufe. Von diesen Unterprogrammen werden weitere Prozeduren aufgerufen usw. Im Programm wird zunächst das Resource-File geladen, das für die GEM-Elemente unerläßlich ist. Bitte beachten Sie dabei unbedingt den Pfadnamen, besonders, wenn Sie das Programm auf andere Disketten und eventuell in Ordner kopieren!

Sie wissen ja, daß das Resource-File mit dem Resource-Construction-Set entwickelt wurde. Bei Bedarf können Sie es natürlich mit Ihrem RCS2.PRG einladen und eventuell sogar ändern (siehe auch Kapitel 8). Bevor Sie das Resource ändern, sollten Sie sich jedoch darüber im klaren sein, daß das Programm 6_4.GFA mit dem geänderten File mitunter nichts mehr anfangen kann. Wir empfehlen daher, das Resource unverändert zu lassen oder höchstens das Aussehen der Objekte zu ändern. Wenn sich dabei die Numerierung der Objekte ändert, müssen Sie das ebenfalls geänderte ASCII-File mit der Extension ».LST« in unser Programm (Prozedur »resource_variablen«) übernehmen.

```
' ------------------------------------------------- UNTERPROGRAMME
PROCEDURE resource_laden(rsc_pfad$)
  IF EXIST(rsc_pfad$)
    OPEN "I",#1,rsc_pfad$
    rsc_laenge%=256*(INT(LOF(#1)/256)+1)
    CLOSE #1
    flag=RSRC_LOAD(rsc_pfad$)
    IF flag=0
      ALERT 3,"Fehler beim Laden|der Resource-Datei.",1,"STOP",var
      END
    ENDIF
```

```
    RESERVE FRE(0)-rsc_laenge%
  ELSE
    ALERT 3,"Ich kann die|Resource-Datei
             nicht finden.",1,"STOP",var
    END
  ENDIF
RETURN
```

In der Prozedur »resource_variablen« werden die Objekt- und Baumnummern des Resource-Files in Variablen übernommen. Dieser Programmteil wurde vom Resource-Construction-Set als ASCII-File auf Diskette gespeichert und anschließend von uns in dieses GFA-Programm übernommen (mit MERGE).

Daran anschließend folgt die Prozedur »variablen«, in der diverse andere im Programm benötigte Variablen angelegt werden. In der Prozedur »vorbereitungen« wird Speicherplatz für den Objektspeicher reserviert und der Bildschirm vorbereitet.

Dann folgt die Prozedur »menu«, die von nun an die Hauptprogrammschleife bildet. Hier wird die Menüzeile kontrolliert und in die entsprechenden Unterprogramme verzweigt.

```
> PROCEDURE resource_variablen
> PROCEDURE variablen
> PROCEDURE vorbereitungen              !Bildschirm u. Speicher vorbereiten
PROCEDURE menu
  REPEAT
    GET 0,0,639,20,menuzeile$           !Hintergrund retten
    teste_ob_anzahl                     !Für Pull-down-Menü "Objekte"
    ~MENU_BAR(adresse1,1)
    ~EVNT_MESAG(0)                      !Menü abfragen...
    ~MENU_TNORMAL(adresse1,MENU(4),1)
    ~MENU_BAR(adresse1,0)
    PUT 0,0,menuzeile$
    SELECT MENU(4)                      !... und auswerten
  CASE titel1&
      SELECT MENU(5)
      CASE info&
        info
      ENDSELECT
      CASE titel2&
      SELECT MENU(5)
      CASE olade&
        lade_objektdatei
      CASE ospei&
        speicher_objektdatei
      CASE omerge&
        merge_objektdatei
```

```
        CASE bloesch&
          zeichne_objekte
        CASE blade&
          bild_laden
        CASE bspei&
          bild_speichern
        CASE ende&
          ende
        ENDSELECT
      CASE titel3&
        SELECT MENU(5)
        CASE neuob&
          neue_objekte
        CASE zeiob&
          objekte_bearbeiten
        CASE loeob&
          objekt_loeschen
        CASE infoob&
          objekt_speicher_info
        ENDSELECT
      CASE titel4&
        SELECT MENU(5)
        CASE paraline&
          parameter_line
        CASE parafill&
          parameter_fill
        CASE paratext&
          parameter_text
        ENDSELECT
      ENDSELECT
    UNTIL 0
  RETURN
```

Die folgenden Prozeduren werden bei Bedarf aufgerufen. Auch hier sind einige »eingeklappt« gekennzeichnet, da der Abdruck in diesem Buch zu umfangreich wäre.

In »lade_objektdatei« kann eine zuvor gespeicherte Objektdatei wieder von Diskette eingeladen werden. Sie sehen, wir überprüfen zunächst den Umfang der Datei daraufhin, ob er überhaupt in unseren Objektspeicher paßt. Wenn ja, wird abgefragt, ob die Datei geladen werden soll oder nicht. Anschließend werden die Objekte dargestellt (Prozedur »zeichne_objekte«).

```
> PROCEDURE info                          !Infobox darstellen
PROCEDURE lade_objektdatei
  FILESELECT DIR$(0)+"\*.*","",objekt_name$
  IF objekt_name$="\" OR objekt_name$=""
  ELSE
    IF EXIST(objekt_name$)
      OPEN "I",#1,objekt_name$
      IF LOF(#1)>laenge%
        ALERT 1,"Diese Datei ist zu|umfangreich.",1,"WEITER",var
      ELSE
        ob_anz%=INP(#1)*256+INP(#1)
        ALERT 1,"Diese Objektdatei enthält|"+STR$
                 (ob_anz%)+" Objekte und ist|"+STR$(LOF(#1)-2)+
                 " Bytes lang.|Datei laden?",1,"JA|NEIN",var
        IF var=1
          BGET #1,ad%,LOF(#1)-2           !Kompletten Block laden...
          ob_ad%=ad%
          zeichne_objekte                 !... und Objekte zeichnen
        ENDIF
      ENDIF
      CLOSE #1
    ENDIF
  ENDIF
RETURN
```

Die folgende Prozedur dient zum Abspeichern des Objektspeichers auf Diskette. Wir laden und speichern hierbei jeweils einen zusammenhängenden Speicherblock mit »BPUT« und »BGET«. Das hat den Vorteil, daß die Speicherung besonders schnell vonstatten geht.

In der Prozedur »merge« kann zu einem bestehenden Objektspeicher ein weiterer hinzugeladen werden.

```
PROCEDURE speicher_objektdatei
  FILESELECT DIR$(0)+"\*.*","",objekt_name$
  IF objekt_name$="\" OR objekt_name$=""
  ELSE
    OPEN "O",#1,objekt_name$
    OUT #1,ob_anz%\256                    !Objektanzahl speichern
    OUT #1,ob_anz%-ob_anz%\256
    BPUT #1,ad%,ob_ad%-ad%                !Kompletten Block speichern
    CLOSE #1
  ENDIF
RETURN
PROCEDURE merge_objektdatei
```

In unserem Programm gibt es zwei Arten von Bildern: die gezeichneten Objekte und den ganz normalen Bildschirmspeicher. Natürlich werden auch die Objekte im Bildschirmspeicher gezeichnet, allerdings kann, bevor die Objektdarstellung beginnt, ein beliebiges Bild in den Bildschirmspeicher geladen werden – quasi als »Hintergrund«. Genauso können Sie neben der reinen Objektdatei den Inhalt des Bildschirmspeichers auf Diskette speichern. Die folgenden Prozeduren ermöglichen das.

```
PROCEDURE bild_laden                          !Hintergrundbild laden
PROCEDURE bild_speichern                      !Bildspeicher abspeichern
```

Die Prozedur »ende« muß, wie der Name schon sagt, am Programmende aufgerufen werden, um den reservierten Speicher wieder freizugeben.

```
PROCEDURE ende
  ALERT 2,"Möchten Sie das Programm|wirklich beenden?",1,"JA|NEIN",var
  IF var=1
    ~MENU_BAR(adresse1,0)
    ~RSRC_FREE()
    ~MFREE(ad%)
    RESERVE FRE(0)+(INT(laenge%/256)+1)*256+rsc_laenge%
    END
  ENDIF
RETURN
```

Die folgende Prozedur wird sehr häufig im Programm aufgerufen. Sie wertet den Objektspeicher aus und ruft die Zeichenroutinen auf, die die Objekte dann schließlich darstellen.

```
PROCEDURE zeichne_objekte                     !Objektspeicher auswerten
  CLS
  IF ob_anz%>0
    ob_zei%=ad%
    FOR i%=1 TO ob_anz%
      ob_code%=DPEEK(ob_zei%)
      ON ob_code% GOSUB g1,g2,g3,g4,g5,g6,g7,g8,g9,g10,g11,g12
      ADD ob_zei%,ob_lae%(ob_code%)
    NEXT i%
  ELSE
    ob_zei%=ad%
  ENDIF
  ob_ad%=ob_zei%                              !Eingabeadresse aktualisieren
RETURN
```

»neue_objekte« stellt zunächst eine Dialogbox auf dem Bildschirm dar, in der Sie auswählen können, welches Objekt Sie zeichnen möchten. Das Ergebnis der Dialogbox (angeklicktes Feld) wird dann ausgewertet und die entsprechende Zeichenroutine aufgerufen. Wir haben hier übrigens nicht alle Zeichenroutinen abgedruckt. Wie Sie sich

denken können, funktionieren die Routinen alle ähnlich. Die Maus wird abgefragt (Maustasten und Mausposition) und daraus werden die Koordinaten der Objekte errechnet. Ein Druck der rechten Maustaste beendet übrigens in allen Zeichenroutinen die Eingabe. Nach der Definition der einzelnen Objekte werden alle Koordinatenwerte der jeweiligen Objekte in den Objektspeicher gespeichert.

```
PROCEDURE neue_objekte                    !Neue Objekte bilden
  REPEAT
    zeig_dialogbox(adresse3,obj%)         !Dialogbox abfragen ...
    DEFMOUSE 5
    DEFLINE stillinie%,dick%,anf%,end%
    DEFFILL farbe%,stilfuell%,muster%
    DEFTEXT farbetext%,stiltext%,richtung%,hoehe%
    SELECT obj%                           !... und in die Routinen verzweigen
    CASE g1&
      g1_n
    CASE g2&
      g2_n
    CASE g3&
      g3_n
    CASE g4&
      g4_n
    CASE g5&
      g5_n
    CASE g6&
      g6_n
    CASE g7&
      g7_n
    CASE g8&
      g8_n
    CASE g9&
      g9_n
    CASE g10&
      g10_n
    CASE g11&
      g11_n
    CASE g12&
      g12_n
    ENDSELECT
    DEFMOUSE 0
  UNTIL obj%=zurueck&
RETURN
```

Mit »objekt_loeschen« können einzelne oder alle Objekte aus dem Speicher und dem Bildschirm entfernt werden. Dazu wird auch wieder eine Dialogbox abgefragt und entsprechend gehandelt.

Als nächste Routine folgt »objekte_bearbeiten«. In einer Dialogbox kann die Nummer des Objektes gewählt werden, das auf dem Bildschirm durch Blinken hervorgehoben werden soll. Wahlweise ist es auch möglich, alle Objekte der Reihe nach zum Blinken zu bringen (weiterschalten mit der linken Maustaste). Wird dann die rechte Maustaste gedrückt, so erscheint eine Dialogbox, die alle wichtigen Informationen zum gewünschten Objekt enthält. Diese Daten können nun geändert werden. Nach Verlassen der Dialogbox wird das Objekt mit den veränderten Daten gezeichnet.

```
PROCEDURE objekt_loeschen
  CHAR{{OB_SPEC(adresse4,loeschnr&)}}=STR$(ob_anz%) !Nr. eintragen
  zeig_dialogbox(adresse4,rueck%)
  OB_STATE(adresse4,weiter4&)=0
  IF rueck%=loeschal&                          !Auswerten
    ALERT 2,"Wirklich ALLE|Objekte löschen?",1,"JA|Oh, nein",var
    IF var=1
      ob_ad%=ad%                               !Zeiger aktualisieren
      ob_anz%=0
      CLS
    ENDIF
  ENDIF
  IF rueck%=loeschei&                          !Nur ein Objekt löschen
    ob_loe%=MAX(0,VAL(CHAR{{OB_SPEC(adresse4,loeschnr&)}}))
    IF ob_loe%<=ob_anz%
      IF ob_loe%<ob_anz%
        ob_zei%=ad%
        FOR i%=1 TO ob_loe%                    !Adresse des Objektes ermitteln
          ob_code%=DPEEK(ob_zei%)
          SELECT ob_code%
          CASE 1 TO 10 !EINFACHE OBJEKTE
            l_ob%=ob_lae%(ob_code%)
          CASE 11 !FREI FÜR ERWEITERUNGEN
            l_ob%=DPEEK(ob_zei%+14)*4-4+ob_lae%(ob_code%)+2
          CASE 12 !TEXTOBJEKT
            string$=CHAR{ob_zei%+ob_lae%(ob_code%)}
            l_ob%=LEN(string$)+1+ob_lae%(ob_code%)
            l_ob%=l_ob%-ODD(l_ob%)
          ENDSELECT
          ADD ob_zei%,l_ob%
        NEXT i%
        ' ---- SPEICHERINHALT ÜBER DEN GELÖSCHTEN BEREICH SCHIEBEN
```

```
        BMOVE ob_zei%,ob_zei%-1_ob%,ad%+laenge%-ob_zei%
      ENDIF
      DEC ob_anz%                         !Ein Objekt weniger
      zeichne_objekte                     !Alles neu zeichnen
    ENDIF
  ENDIF
RETURN
PROCEDURE objekte_bearbeiten
  CHAR{{OB_SPEC(adresse5,zeignr&)}}=STR$(ob_anz%)
  zeig_dialogbox(adresse5,rueck%)
  OB_STATE(adresse5,weiter5&)=0
  zeignr%=MIN(VAL(CHAR{{OB_SPEC(adresse5,zeignr&)}}),ob_anz%+1)
  IF rueck%<>weiter5&                   !Ergebnis der Dialogbox auswerten
    IF rueck%=zeigobj&
      objekt_blink(zeignr%,maust&)
    ENDIF
    IF rueck%=objausw&
      CLR zeignr%
      REPEAT
        INC zeignr%
        objekt_blink(zeignr%,maust&)
        IF maust&=1
          REPEAT
          UNTIL MOUSEK<>1
        ENDIF
      UNTIL zeignr%=ob_anz% OR maust&=2
    ENDIF
    IF maust&=2
      ob_info_zeigen(zeignr%)             !Infobox darstellen
      zeichne_objekte
    ENDIF
  ENDIF
RETURN
```

Es folgen Prozeduren, die eine Information über den beanspruchten Speicherplatz geben (objekt_speicher_info) bzw. weitere Dialogboxen darstellen, in denen die Zeichenparameter eingestellt werden können. Einige weitere Prozeduren werden für kleinere Aufgaben genutzt.

```
> PROCEDURE objekt_speicher_info
> PROCEDURE parameter_line
> PROCEDURE parameter_fill
> PROCEDURE parameter_text
'
```

```
' ------------------------------------------ WEITERE HILFSROUTINEN
'
> PROCEDURE teste_ob_anzahl
> PROCEDURE werte_holen(peekadresse%,anzahl%)
> PROCEDURE werte_poken(adr%,we1%,we2%,we3%,we4%,we5%,we6%,we7%,we8%)
> PROCEDURE objekt_blink(blinknr%,VAR maustaste&)
> PROCEDURE ob_info_zeigen(infonr%)
RETURN
```

In der folgenden Routine wird die Anfangsadresse eines Objektes im Objektspeicher ermittelt. Bei zwei Objekttypen (Type 11 und 12) ist der benötigte Speicherplatz unterschiedlich groß und muß also von Objekt zu Objekt einzeln ermittelt werden, was die Adreßberechnung aufwendig macht.

```
PROCEDURE adresse_ermitteln(objektnummer%,VAR objektadresse%)
  objektadresse%=ad%
  FOR i%=1 TO objektnummer%
    ob_code%=DPEEK(objektadresse%)
    SELECT ob_code%
    CASE 1 TO 10
      l_ob%=ob_lae%(ob_code%)
    CASE 11
      l_ob%=DPEEK(objektadresse%+14)*4-4+ob_lae%(ob_code%)+2
    CASE 12
      string$=CHAR{objektadresse%+ob_lae%(ob_code%)}
      l_ob%=LEN(string$)+1+ob_lae%(ob_code%)
      l_ob%=l_ob%-ODD(l_ob%)
    ENDSELECT
    ADD objektadresse%,l_ob%
  NEXT i%
  SUB objektadresse%,l_ob%
RETURN
> PROCEDURE mausposition(VAR mausx%,mausy%,maust%)
> PROCEDURE mausposition2(VAR mausx%,mausy%,maust%)
> PROCEDURE speicher_reservieren(laenge_speicher%,VAR adresse_speicher%)
```

»zeig_dialogbox« stellt eine Dialogbox mit der angegebenen Anfangsadresse des Resource-Speichers dar. Die Box wird dann abgefragt und das Ergebnis, also die Nummer des angeklickten Feldes, zurückgegeben.

Im weiteren Verlauf dieses Listings sehen Sie noch einige Zeichenroutinen. Die Routinen mit dem »n« am Ende des Namens dienen dazu, neue Objekte zu bilden. Die anderen Routinen zeichnen Objekte aus dem Objektspeicher. Dazu müssen dann natürlich jeweils die Daten aus dem Speicher ausgelesen werden, die sonst mit der Maus gebildet werden.

Die Prozedur »g11« stellt eine »Freihandlinie« dar. Dabei werden die definierten Ecken des Objektes mit Linien verbunden.

```
PROCEDURE zeig_dialogbox(boxadresse,VAR rueckgabe_wert%)
  SGET bildschirm$
  ~FORM_CENTER(boxadresse,x%,y%,w%,h%)
  ~FORM_DIAL(0,0,0,0,0,x%,y%,w%,h%)
  ~FORM_DIAL(1,0,0,0,0,x%,y%,w%,h%)
  ~OBJC_DRAW(boxadresse,0,8,x%,y%,w%,h%)
  rueckgabe_wert%=FORM_DO(boxadresse,0)
  ~FORM_CENTER(boxadresse,x%,y%,w%,h%)
  ~FORM_DIAL(2,0,0,0,0,x%,y%,w%,h%)
  ~FORM_DIAL(3,0,0,0,0,x%,y%,w%,h%)
  SPUT bildschirm$
RETURN
> PROCEDURE grafik_laden(bild_name$,bild_adresse%,filenummer%)
'
' -------------------------             ZEICHENROUTINEN FÜR JEWEILS EIN OBJEKT
'
PROCEDURE g1                            !Draw
  werte_holen(ob_zei%+2,6)
  DEFLINE we%(1),we%(2),we%(3),we%(4)
  DRAW we%(5),we%(6)
RETURN
PROCEDURE g1_n
  REPEAT
    mausposition(mx1%,my1%,mt1%)
    IF mt1%=1
      DRAW mx1%,my1%
      DPOKE ob_ad%,1 !OB_CODE
      werte_poken(ob_ad%+2,stillinie%,dick%,anf%,end%,mx1%,my1%,0,0)
      ADD ob_ad%,14
      INC ob_anz%
      WHILE MOUSEK=1
      WEND
    ENDIF
  UNTIL mt1%=2
RETURN
> PROCEDURE g2                          !Line
> PROCEDURE g2_n
> PROCEDURE g3                          !Box
> PROCEDURE g3_n
> PROCEDURE g4                          !Rbox
> PROCEDURE g4_n
```

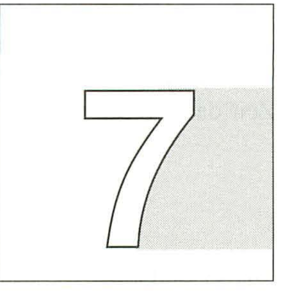
```
> PROCEDURE g5                          !Circle
> PROCEDURE g5_n
> PROCEDURE g6                          !Ellipse
> PROCEDURE g6_n
> PROCEDURE g7                          !Pbox
> PROCEDURE g7_n
> PROCEDURE g8                          !Prbox
> PROCEDURE g8_n
> PROCEDURE g9                          !Pcircle
> PROCEDURE g9_n
> PROCEDURE g10                         !Pellipse
> PROCEDURE g10_n
PROCEDURE g11                           !Freihand
  werte_holen(ob_zei%+2,7)
  ob_zei_alt%=ob_zei%+2
  ADD ob_zei%,16
  DEFLINE we%(1),we%(2),we%(3),we%(4)
  DRAW we%(5),we%(6) TO we%(5),we%(6)
  xa%=we%(5)
  ya%=we%(6)
  FOR j%=1 TO we%(7)
    xe%=xa%+DPEEK(ob_zei%)
    ye%=ya%+DPEEK(ob_zei%+2)
    ADD ob_zei%,4
    DRAW xa%,ya% TO xe%,ye%
    xa%=xe%
    ya%=ye%
  NEXT j%
  ob_zei%=ob_zei_alt%+we%(7)*4-4
RETURN
PROCEDURE g11_n
  CLR ecken%
  mausposition(xa%,ya%,mt%)             !Erster Punkt
  IF mt%=1
    INC ob_anz%
    DPOKE ob_ad%,11 !OB_CODE
    werte_poken(ob_ad%+2,stillinie%,dick%,anf%,end%,xa%,ya%,0,0)
    eckenadr%=ob_ad%+14
    ADD ob_ad%,16
    xe%=xa%
    ye%=ya%
    INC ecken%
    REPEAT
```

```
      REPEAT
        mausposition2(xe1%,ye1%,mt2%)
      UNTIL xe1%<>xa% OR ye1%<>ya%
      dx%=xe1%-xa%
      dy%=ye1%-ya%
      LINE xa%,ya%,xa%+dx%,ya%+dy%
      xa%=xa%+dx%
      ya%=ya%+dy%
      DPOKE ob_ad%,dx%
      DPOKE ob_ad%+2,dy%
      ADD ob_ad%,4
      INC ecken%
    UNTIL mt2%=0
    DPOKE eckenadr%,ecken%
  ENDIF
RETURN
> PROCEDURE g12                         !Text
> PROCEDURE g12_n
```

Damit ist dieses sehr umfangreiche Listing beendet. Bitte haben Sie Verständnis dafür, daß wir es nicht in allen Punkten ausführlich beschreiben können, das wäre wirklich zu umfangreich!

Sie sehen auch an diesem Programm, daß man mit GFA-Basic 3.0 sehr wohl in der Lage ist, leistungsfähige Programme zu erstellen. Kompiliert werden diese dann sehr schnell. Wir hoffen, daß Sie durch unser Buch eine Hilfestellung bei der Erstellung Ihrer eigenen Programme erhalten. Sie können viele unserer Prozeduren ohne Änderung in Ihre eigenen Programme übernehmen (z.B. »grafik_laden« oder »zeig_dialogbox« usw.).

Im nächsten Abschnitt beschreiben wir einige besondere Funktionen des GFA-Basic, mit denen man z.B. neue Zeichensätze nutzen kann.



7 Fortgeschrittene Programmierung

In den vorausgehenden Kapiteln zeigten wir Ihnen einige interessante Anwendungen für den Atari ST unter GFA-Basic. Natürlich konnte dabei nicht auf jeden Befehl einzeln ausführlich eingegangen werden, das Buch wäre sonst ein riesiger und teurer Wälzer, was Sie und wir nicht wollen.

In diesem Kapitel, »Fortgeschrittene Programmierung«, bringen wir Ihnen in lockerer Reihenfolge einige Besonderheiten Ihres Rechners und des GFA-Basic näher. Zunächst betrachten wir das GDOS, dann den Befehl RC_COPY und die Möglichkeiten, die er bietet, und dann geht es um den Aufruf von Maschinensprachprogrammen aus dem Basic heraus. Dort finden Sie auch eine Maschinenroutine, die den Bildschirminhalt blitzschnell spiegelt. Wir hoffen, auch bei dieser Auswahl wieder Ihren Geschmack getroffen zu haben.

7.1 Erweiterte Grafikfunktionen durch GDOS

Den meisten von Ihnen wird die Abkürzung GDOS nichts sagen. Es handelt sich dabei um das *Graphics Device Operating System*, eine Erweiterung des VDI (*Virtual Device Interface*). An dieser Beschreibung erkennen Sie schon, daß diese Erweiterung nur bei Bedarf genutzt werden kann. Genaugenommen muß die VDI-Erweiterung nachgeladen werden, bevor die Funktionen dann in Programmen genutzt werden können. Doch wozu dient nun dieses GDOS?

Wie Sie wissen, bietet das VDI hauptsächlich Grafikroutinen. Hier sind die Grundroutinen enthalten, von denen das AES reichlich Gebrauch macht. Im Grunde genommen liefert das VDI aber noch mehr: die Gerätetreiber. Hierdurch ist es möglich, Grafikausgaben auf alle möglichen Geräte zu leiten. Natürlich ist es nicht möglich und auch nicht sinnvoll, sämtliche Gerätetreiber ständig im Speicher des ST zu behalten. Diese müssen also bei Bedarf nachgeladen werden. Das sollte eigentlich das VDI ermöglichen. Bedauerlicherweise kann es das aber nicht, denn das VDI Ihres ST ist »nicht ganz vollständig«. Es fehlt die Möglichkeit, Gerätetreiber nachzuladen.

Diesen Mangel (unter anderen) behebt das GDOS. Dadurch ist es zum Beispiel möglich, einen Laser-Drucker oder einen Plotter anzusteuern, und zwar genauso, wie Sie den Bildschirm ansteuern.

Glücklicherweise ist das GDOS kostenlos bei Atari zu erhalten. Sie sollten sich allerdings vorher kurz bei Atari oder Ihrem Händler erkundigen, wie zur Zeit das GDOS geliefert werden kann.

7.1.1 Neue Zeichensätze

Sind Sie nun glücklicher Besitzer des Programms GDOS, können Sie es zum Beispiel dazu nutzen, Grafikausgaben auf Ihren Drucker zu leiten. Sie können aber zum Beispiel auch neue Zeichensätze einladen und in Ihren Programmen nutzen. Um dieses Thema soll es im nächsten Abschnitt gehen. Wir zeigen Ihnen dazu weniger die Anwendung der neuen Zeichensätze, denn die ist recht einfach, sondern vielmehr ihre Erstellung. Dazu liefern wir auch gleich ein Programm mit (Programm 7_1.GFA), mit dem Sie beliebige Zeichensätze ändern oder sogar neu erstellen können.

Bevor wir dieses Programm beschreiben, möchten wir einige Worte zu Zeichensätzen und deren Anwendung bringen. Der Atari ST liefert »von Haus aus« schon drei Zeichensätze mit. Diese werden natürlich auch ständig genutzt, um Texte auf dem Bildschirm darzustellen. Der erste Zeichensatz enthält 8x16 Pixel große Zeichen und wird hauptsächlich im hochauflösenden Modus eingesetzt. Für den Farbbetrieb gibt es den 8x8-Pixel-Zeichensatz. Der dritte im Bunde enthält 6x6 Pixel kleine Zeichen und wird zum Beispiel für die Desktop-Piktogramme verwendet. Somit sind die Möglichkeiten der Textdarstellung also ziemlich begrenzt. Mit neu hinzugeladenen Zeichensätzen dagegen öffnet sich Ihnen ein Tor zu ganz neuen Dimensionen. So sind zum Beispiel die Desktop-Publishing-Programme selbstverständlich auf neue Zeichensätze angewiesen.

Die Zeichensätze müssen sich außerdem nicht ausschließlich auf Ziffern und Buchstaben beschränken, sondern können Symbole oder sogar kleine Zeichnungen enthalten. Die Ausgabemöglichkeiten der Zeichen durch das GEM bieten zudem auch noch diverse Beeinflussungen, wie zum Beispiel die kursive, helle oder unterstrichene Ausgabe auf das angesteuerte Gerät. Ihnen stehen also viele Möglichkeiten offen. Allerdings benötigen Sie dazu das GDOS, das beim Booten des Rechners eingeladen wird.

Um GDOS zu nutzen, muß das Programm im AUTO-Ordner Ihrer Boot-Diskette enthalten sein. Außerdem wird eine Steuerdatei benötigt, die Angaben über die verwendeten Gerätetreiber und Zeichensätze enthält. Bei dieser Steuerdatei handelt es sich um eine ASCII-Datei, die der Reihe nach die verschiedenen Gerätetreiber, auch die Bildschirmtreiber, und die jeweils benötigten Zeichensätze angibt. Zum GDOS ist eine derartige Datei natürlich enthalten. Sie können das File selbstverständlich auch ändern, und zwar, indem Sie es mit einer Textverarbeitung einladen und als ASCII-Datei nach der Änderung wieder anspeichern.

Die Steuerdatei besitzt den Namen ASSIGN.SYS. Wir zeigen Ihnen einen kleinen Ausschnitt aus einer derartigen Datei:

```
PATH = C:\GEMSYS\
01p screen.sys
ATSS10.FNT
ATTP10.FNT
02p screen.sys
ATSS10.FNT
ATTP10.FNT
03p screen.sys
.
.
.
31 META.SYS
ATSS10MF.FNT
ATSS12MF.FNT
ATSS18MF.FNT
```

Mit »PATH...« wird der Pfadname für die nachzuladenden Zeichensätze und Gerätetreiber angegeben. Diese Dateien sollten also alle im selben Inhaltsverzeichnis (Ordner) Ihrer Diskette sein. Weitere Informationen zu ASSIGN.SYS finden Sie in der speziellen GEM-Literatur und in Ihrer GFA-Basic-Anleitung.

Wir kommen nun zum Aufbau der Zeichensätze. Jedem Zeichensatz ist ein sogenannter *Font-Header* vorangestellt, der diverse Informationen über den Zeichensatz enthält. Die Daten der Zeichen selbst befinden sich in einer Zeichenmatrix, die ein besonderes Format hat. Die Matrix besteht aus einer bestimmten Anzahl *Rasterzeilen* (die Anzahl ist im Font-Header enthalten). Innerhalb dieser Rasterzeilen sind nun die Zeichen nebeneinander gespeichert. Jedes Zeichen kann dabei eine unterschiedliche Breite annehmen (siehe auch Bild 7.1). Um nun herauszubekommen, an welcher Stelle welches Zeichen beginnt, enthält der Zeichensatz noch eine weitere Tabelle (Zeichen-Offset-Tabelle), die für jedes Zeichen angibt, an welcher Spalte des Definitionsrasters es beginnt.

Diese Art der Zeichensatzdefinierung zeichnet sich dadurch aus, daß sie neben der großen Flexibilität auch noch sehr speicherplatzsparend ist.

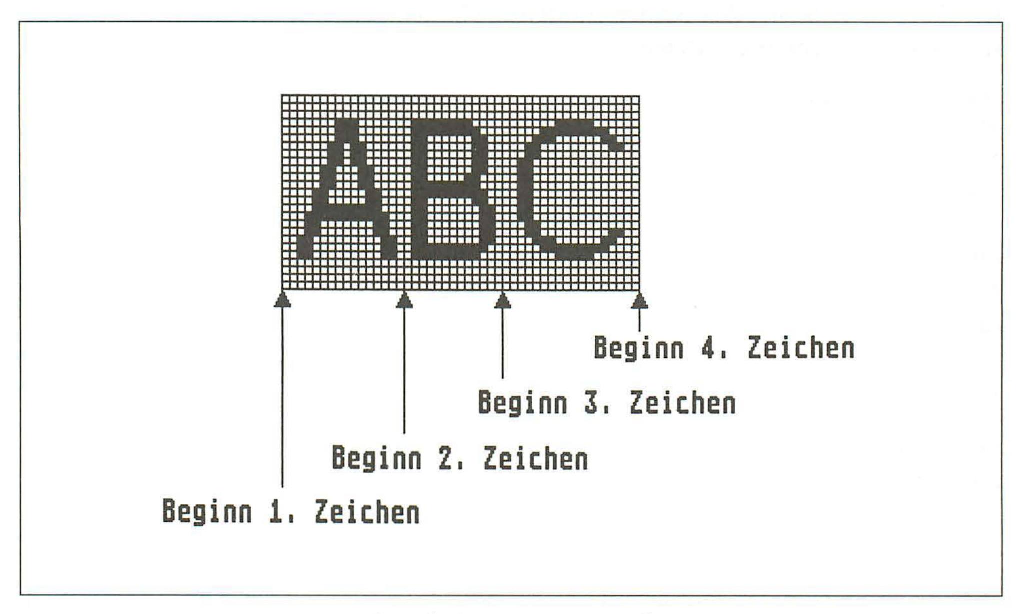


Bild 7.1: Ausschnitt aus einem Zeichenraster

Die einzelnen Daten des Font-Headers haben folgende Bedeutung:

Byte	Type	Bezeichnung
0,1	Word	Zeichensatznummer
2,3	Word	Größe der Zeichen in $^1/_{72}$ Inch (für Drucker)
4,35	String	Maximal 32 Zeichen langer Zeichensatzname
36,37	Word	Kleinster ASCII-Wert im Zeichensatz
38,39	Word	Höchster ASCII-Wert im Zeichensatz
40,41	Word	Abstand zwischen Topline und Baseline
42,43	Word	Abstand zwischen Ascentline und Baseline
44,45	Word	Abstand zwischen Halfline und Baseline
46,47	Word	Abstand zwischen Descentline und Baseline
48,49	Word	Abstand zwischen Bottomline und Baseline (siehe dazu auch das Bild 7.2)
50,51	Word	Breite des breitesten Zeichens
52,53	Word	Breite der breitesten Zeichenzelle
54,55	Word	Linker Offset für Kursiv
56,57	Word	Rechter Offset für Kursiv

Byte	Type	Bezeichnung
58,59	Word	Anzahl der zusätzlichen Pixel bei Fettschrift
60,61	Word	Punktanzahl des Unterstreichstriches
62,63	Word	Maske für helle Schrift
64,65	Word	Maske für Kursivschrift
66,67	Word	Modus
68-71	Long	Zeiger auf die Horizontal-Offset-Tabelle
72-75	Long	Zeiger auf die Zeichen-Offset-Tabelle
76-79	Long	Zeiger auf die Zeichenmatrix
80,81	Word	Länge einer Zeile der Zeichenmatrix in Bytes
82,83	Word	Anzahl der Rasterzeilen
84-87	Long	Zeiger auf den nächsten Zeichensatz

Zu dieser Tabelle noch einige Anmerkungen. In den Bytes 40 bis 49 befinden sich einige Angaben über den Aufbau der sogenannten Zeichenzellen. In Bild 7.2 sehen Sie, was mit den Bezeichnungen gemeint ist. Für Sie sind diese Angaben allerdings nicht so sehr wichtig. Es muß nur eines beachtet werden: Der Abstand zwischen Topline und Baseline ist entscheidend dafür, welcher Zeichensatz bei einer eingestellten Textgröße (DEFTEXT farbe,stil,winkel,hoehe...) genutzt wird.

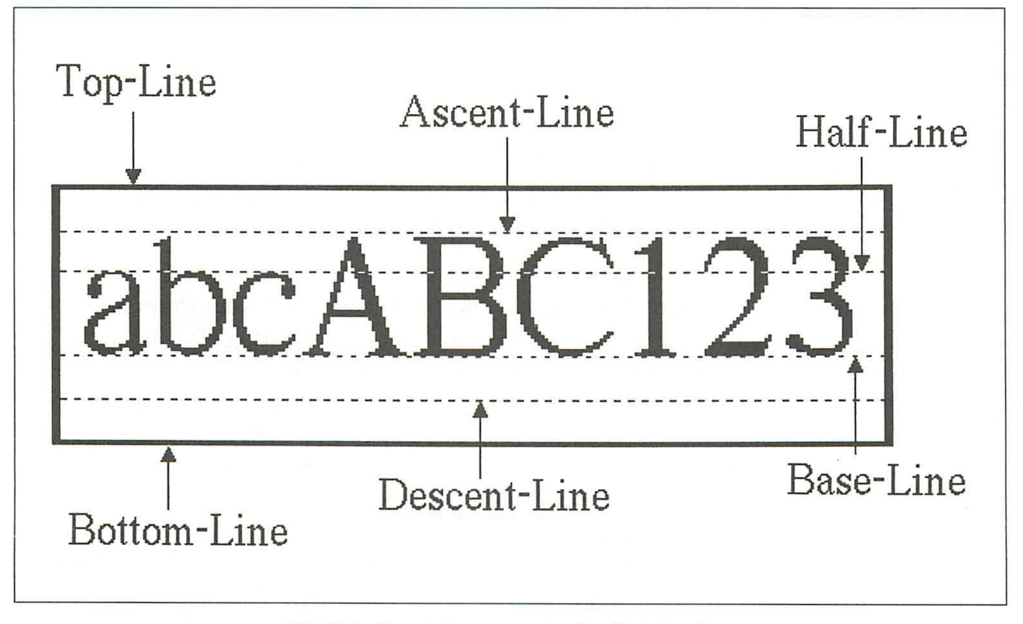


Bild 7.2: Bezeichnungen in der Zeichenbox

Die Angaben in den Bytes 40 bis 49 sind für einige VDI-Routinen sehr wichtig. Unser GFA-Basic bietet allerdings lediglich den Befehl »TEXT«, der eigentlich nicht alle Darstellungs- und Beeinflussungsmöglichkeiten unterstützt (natürlich können alle VDI-Funktionen mit »VDISYS« aufgerufen werden, doch dazu mehr an anderer Stelle in diesem Buch). Die zusätzlichen VDI-Funktionen sind aber für die normale Arbeit mit dem ST nicht erforderlich.

Um weitere Informationen über Zeichensätze zu erhalten, empfehlen wir Ihnen, spezielle GEM-Literatur zu lesen. Für den Anfang sollten Sie einfach schon vorhandene Zeichensätze ändern (vorher unbedingt Sicherheitskopien anfertigen). Lassen Sie dabei einfach die Parameter unverändert, dann dürften keine Schwierigkeiten mit dem Zeichensatz auftreten.

Die Masken für die helle Schrift und die Kursivschrift enthalten als üblichen Wert &H5555 (hexadezimal, entspricht 21845 dezimal). Dadurch wird bei der hellen Schrift jeder zweite Punkt gelöscht.

Der Modus in den Bytes 66 und 67 enthält folgende Angaben:

Bit	**Format**
Bit 0:	1 = Systemfont, 0 = anderer Font
Bit 1:	1 = Horizontal-Offset-Tabelle vorhanden
Bit 2:	1 = Worte und Langworte im Intel-Format (Lowbyte, Highbyte), 0 = MC68000-Format (Highbyte, Lowbyte)
Bit 3:	1 = Alle Zeichen im Font haben die gleiche Breite

Alle Zeichensätze, die wir bisher gesehen haben, hatten als Modus den Wert 0. In unserem Zeichensatzeditor haben wir es daher nicht vorgesehen, daß der Modus geändert werden kann. Er wird bei jedem neuen Zeichensatz auf 0 gesetzt.

Die *Horizontal-Offset*-Tabelle wird nur selten benutzt. Wir haben daher auf das Anlegen einer solchen Tabelle in unserem Programm verzichtet. Dadurch kann das Programm allerdings auch keine Zeichensätze einwandfrei auswerten, die eine derartige Tabelle enthalten. Es handelt sich bei der Horizontal-Tabelle um eine zweite *Zeichen-Offset*-Tabelle, die es ermöglicht, zum Beispiel vor einem Zeichen zusätzlichen Leerraum auszugeben oder Teile eines Zeichens nicht auszugeben. Man kann so aus einem Zeichensatz mit gleichmäßig großen (breiten) Zeichen einen Proportional-Zeichensatz machen. Allerdings waren alle Zeichensätze, die wir bisher sahen, Proportional-Zeichensätze (mit unterschiedlich breiten Zeichen), in denen die Horizontal-Tabelle nicht vorhanden war. Als Wert steht dann im Langwort ab Byte 68 derselbe Wert wie im Langwort ab Byte 72 (Zeiger auf die Zeichen-Offset-Tabelle). In unserem Programm können diese Parameter (Zeichen-Offset-Tabelle und Horizontal-Offset-Tabelle und Modus) nicht eingestellt werden. Das Programm übernimmt sich nämlich selbst.

Wenn Sie sich die 88 Byte des Font-Headers ansehen, werden Sie erkennen, wie flexibel so ein GEM-Zeichensatz ist. Allerdings ist er dadurch auch ziemlich kompliziert. Folglich wird es auch kompliziert sein, einen neuen Zeichensatz zu entwickeln. Aus diesem Grund gibt es für diesen Zweck auch nur sehr wenige Programme. Das war für uns die Herausforderung, ein derartiges Programm einmal in GFA-Basic zu schreiben. Wie wir schon angesprochen haben, haben wir einige Abstriche in der Flexibilität unseres Programms gemacht. So ein Projekt kann nämlich auch leicht den Rahmen eines Buches über das GFA-Basic 3.0 sprengen. Auf dem Gebiet des GDOS wagen wir uns ja schon ziemlich weit in spezielle Anwendungsgebiete. Wir möchten Ihnen allerdings auch zeigen, was Sie mit dem GFA-Basic alles »anfangen« können und daß es vermutlich kaum einen Bereich gibt, in dem man auf die Vorzüge der Programmiersprache Basic verzichten muß. Durch dieses Buch sollen Sie sehen, daß selbst komplizierte Probleme durchaus gelöst werden können.

Unser Programm heißt »Font-Editor«, und Sie finden es auf der beiliegenden Diskette unter dem Namen 7_1.GFA. Wir drucken es jetzt hier in Teilen ab. Einige Routinen wie das Einladen einer Grafik oder das Anzeigen oder Löschen einer Dialogbox sind übrigens auch in den vorhergehenden Programmen genauso vorhanden und dort schon beschrieben worden. Wir haben diese Routinen daher nur als »eingeklappte« Prozeduren gekennzeichnet.

```
' *****                    PROGRAMM 7.1                        *****
' *****************************************************************
' *****                                                        *****
' *****                     FONTEDITOR                         *****
' *****                FÜR GEM-ZEICHENSÄTZE                    *****
' *****                                                        *****
' *****          WILHELM BESENTHAL UND JENS MUUS               *****
' *****                                                        *****
' *****           MARKT & TECHNIK, MÜNCHEN 1989                *****
' *****                                                        *****
' *****************************************************************
'
'
'
' DER FOLGENDE AUFRUF ENTHÄLT DAS MASCHINENPROGRAMM
' FÜR DAS UP 'SPIEGELN'. IN DER NÄCHSTEN ZEILE KEIN REM EINGEBEN!
INLINE adprg%,238
'
'
resource_laden(CHR$(GEMDOS(25)+65)+":"+DIR$(0)+"\7.GFA\7_1.RSC")
resource_variablen
variablen
menu
```

Zu Programmbeginn ist der Befehl »INLINE« angegeben. Dieser reserviert einen Speicherbereich von hier 238 Byte Länge. Das Besondere an »INLINE« ist, daß der Speicherbereich unmittelbar an dieser Stelle im GFA-Basic-Programm (also im Listing) freigehalten wird. Hier kann man nun beliebige eigene Daten einfügen, die dann auch zusammen mit dem Programm abgespeichert oder eingeladen werden. Das kann man zum Beispiel für Bilder oder auch für Maschinenroutinen nutzen. In unserem Beispiel befindet sich an dieser Stelle eine Maschinenroutine zum Spiegeln von Bildschirmbereichen. Nähere Informationen hierzu finden Sie im Kapitel 7.3.

Es folgen nun schon die diversen Unterprogramme, von denen die ersten die Resource-Datei einladen und wichtige Variablen definieren. Die Prozedur »vorbereitungen« ist die Routine, die den Anwender vor die Entscheidung stellt, entweder einen völlig neuen Zeichensatz zu entwickeln oder aber einen schon vorhandenen einzuladen und zu verändern.

```
> PROCEDURE resource_laden(rsc_pfad$)
> PROCEDURE resource_variablen
> PROCEDURE variablen                         !Wichtige Variablen deklarieren
PROCEDURE vorbereitungen
  IF XBIOS(4)<>2
    ALERT 3,"Programm für die|hohe Bildschirm-
            |Auflösung",1,"ENDE",var
    ende
  ENDIF
  ARRAYFILL links%(),0
  ARRAYFILL rechts%(),0
  ARRAYFILL breite%(),0
  ARRAYFILL z_offs%(),0
  CLS
  SGET eingabebild$
  SGET grafikbild$
  PRINT AT(34,2);"FONT-EDITOR"
  ALERT 2,"Möchten Sie einen neuen|Zeichensatz entwickeln|
          oder einen schon vorhandenen|ändern?",2,"NEU|ÄNDERN",var
  IF var=1                                    !Einen neuen Zeichensatz entwickeln
    speicher_groesse%=40000                   !Kann bei Bedarf geändert werden
    speicher_reservieren(speicher_groesse,laenge%)
    mode%=0
    asc_l%=0
    asc_h%=255
    f_name$="Neu"
    z_name$=""
    rasterzeilen%=14
    hell_maske%=21845
    kursiv_maske%=21845
```

```
    hor_tab%=88
    zei_tab%=88
    OB_STATE(adresse3,d82&)=0            !Rasterzeilen freigeben
    OB_FLAGS(adresse3,d82&)=8
    parameter
    groesse%=rasterzeilen%/2
    zeichen%=asc_l%                      !Das aktuelle Zeichen
  ELSE                 !Vorhandenen Zeichensatz laden und bearbeiten
    FILESELECT lwz$+"\*.*","",name$
    IF RIGHT$(name$)="\" OR name$=""
      ~MENU_BAR(adresse1,0)
      ~RSRC_FREE()
      r%=MFREE(ad%)
      RESERVE FRE(0)+rsc_laenge%
      END
    ELSE
      z_name$=name$
      lwz$=LEFT$(name$,RINSTR(name$,"\")-1)
      OPEN "I",#1,name$
      speicher_groesse%=LOF(#1)
      CLOSE #1
      speicher_reservieren(speicher_groesse%,laenge%)
      BLOAD name$,ad%
      zeichensatz_auswerten
      zeichen%=asc_l%
      OB_STATE(adresse3,d82&)=8          !Rasterzeilen schützen
      OB_FLAGS(adresse3,d82&)=0
      parameter
    ENDIF
  ENDIF
  DEFLINE 1,1
  OB_STATE(adresse3,d82&)=8              !Rasterzeilen schützen
  OB_FLAGS(adresse3,d82&)=0
RETURN
> PROCEDURE bildschirm                   !Eingabebild aufbauen
```

Die Prozedur »menu« stellt die Hauptprogrammschleife dar. Hier werden die Aktionen der Maus überwacht und in die entsprechenden Unterprogramme verzweigt. Wir nutzen hier übrigens »EVNT_MULTI...«, das die Pull-down-Menüs und die linke Maustaste überwacht. An anderer Stelle in diesem Buch gehen wir auf alle Einzelheiten dieser Ereignisüberwachung genau ein. Übrigens geben wir als »message_adr%« den Wert 0 an, was bewirkt, daß GFA-Basic die Ergebnisse der Ereignisüberwachung in die Werte »MENU()« übergibt.

```
PROCEDURE menu
  REPEAT
    vorbereitungen                      !Variablen, Bildschirm usw.
    REPEAT
      rueck%=EVNT_MULTI(&X10010,1,1,1,0,0,0,0,0,0,0,0,0,0,
             message_adr%,0,mx%,my%,mb%,d%,d%,d%)
      IF BTST(rueck%,4)                 !Pull-down-Menü angeklickt?
        ~MENU_TNORMAL(adresse1,MENU(4),1)
        SELECT MENU(4)                  !In die Routinen
        CASE titel1&
          IF MENU(5)=info&
            info
          ENDIF
        CASE titel2&
          IF MENU(5)=speicher&
            zeichen_uebernehmen
            zeichensatz_speichern
            bildschirm
          ENDIF
          IF MENU(5)=neu&
            ALERT 2,"Möchten Sie wirklich|einen neuen
                     Zeichensatz|bearbeiten?",1,"JA|NEIN",var
            IF var=1
              zeichen_uebernehmen
              ~MENU_BAR(adresse1,0)
              ~RSRC_FREE()
              r%=MFREE(ad%)
              RESERVE FRE(0)+laenge%
              neu%=1
            ENDIF
          ENDIF
          IF MENU(5)=ende&
            ende
          ENDIF
        CASE titel3&
          IF MENU(5)=laden&
            bild_laden
          ENDIF
          IF MENU(5)=paramet&
            zeichen_uebernehmen
            parameter
          ENDIF
        ENDSELECT
```

```
ELSE                                  !Kein Pull-down-Menü, also übriger Bildschirm?
  IF mx%>200 AND mx%<295
    IF my%>100 AND my%<115
      grafikbildschirm
    ENDIF
    IF my%>120 AND my%<135
      matrix_loeschen
    ENDIF
    IF my%>140 AND my%<155
      spiegeln_horizontal
    ENDIF
    IF my%>160 AND my%<175
      spiegeln_vertikal
    ENDIF
    IF my%>180 AND my%<195
      matrix_hoch
    ENDIF
    IF my%>200 AND my%<215
      matrix_runter
    ENDIF
    IF my%>220 AND my%<235
      matrix_links
    ENDIF
    IF my%>240 AND my%<255
      matrix_rechts
    ENDIF
  ENDIF
  IF mx%>530 AND mx%<540
    IF my%>26 AND my%<40
      zeichen_schmaler
    ENDIF
    IF my%>46 AND my%<60
      zei_del%=-1
      zeichen_ab
    ENDIF
  ENDIF
  IF mx%>580 AND mx%<590
    IF my%>26 AND my%<40
      zeichen_breiter
    ENDIF
    IF my%>46 AND my%<60
      zei_del%=1
      zeichen_ab
```

```
          ENDIF
        ENDIF
        IF mx%>506 AND mx%<522
          IF my%>46 AND my%<60
            zei_del%=-10
            zeichen_ab
          ENDIF
        ENDIF
        IF mx%>596 AND mx%<612
          IF my%>46 AND my%<60
            zei_del%=10
            zeichen_ab
          ENDIF
        ENDIF
        IF mx%>320 AND mx%<320+breite%(zeichen%)*4
           AND my%>100 AND my%<100+rasterzeilen%*4
          maus_in_matrix
        ENDIF
      ENDIF
    UNTIL neu%=1
    CLR neu%
  UNTIL 0
RETURN
> PROCEDURE info                              !Info-Dialogbox darstellen
```

In »speicher_reservieren« kann eine »anzahl%« Byte Speicherbereich vom Basic-Speicher abgeschnitten werden. Dies geht allerdings nur in 256er-Schritten. Die Funktion berechnet daher die genaue Anzahl der Bytes und gibt diese in »reserviert%« wieder zurück.

```
PROCEDURE speicher_reservieren(anzahl%,VAR reserviert%)
  reserviert%=256*(INT(anzahl%/256)+1)
  RESERVE FRE(0)-reserviert%
  ad%=MALLOC(reserviert%)
  IF ad%=0
    ALERT 3,"Zu wenig Speicherplatz!",1,"ENDE",var
    ende
  ENDIF
RETURN
> PROCEDURE spiegeln_horizontal
> PROCEDURE spiegeln_vertikal
> PROCEDURE spiegeln
> PROCEDURE matrix_hoch                       !Matrixinhalt um 1 Pixel verschieben
> PROCEDURE matrix_runter
> PROCEDURE matrix_links
```

```
> PROCEDURE matrix_rechts
> PROCEDURE parameter_belegen              !Werte in Dialogbox
> PROCEDURE parameter                      !Parameter-Dialogbox
```

Die folgende Prozedur wertet die Angaben der Parameter-Dialogbox aus.

```
PROCEDURE parameter_auswerten !Werte aus Dialogbox
  f_nr%=VAL(CHAR{{OB_SPEC(adresse3,d0&)}})
  rasterzeilen%=MIN(VAL(CHAR{{OB_SPEC(adresse3,d82&)}}),72)
  f_name$=CHAR{{OB_SPEC(adresse3,d4&)}}
  asc_h%=MIN(VAL(CHAR{{OB_SPEC(adresse3,d38&)}}),255)
  asc_l%=MIN(VAL(CHAR{{OB_SPEC(adresse3,d36&)}}),asc_h%)
  zeichen%=MAX(MIN(asc_h%,zeichen%),asc_l%)
  tl_bl%=VAL(CHAR{{OB_SPEC(adresse3,d40&)}})
  al_bl%=VAL(CHAR{{OB_SPEC(adresse3,d42&)}})
  hl_bl%=VAL(CHAR{{OB_SPEC(adresse3,d44&)}})
  dl_bl%=VAL(CHAR{{OB_SPEC(adresse3,d46&)}})
  bl_bl%=VAL(CHAR{{OB_SPEC(adresse3,d48&)}})
  b_max%=VAL(CHAR{{OB_SPEC(adresse3,d50&)}})
  b_box%=VAL(CHAR{{OB_SPEC(adresse3,d52&)}})
  l_offset%=VAL(CHAR{{OB_SPEC(adresse3,d54&)}})
  r_offset%=VAL(CHAR{{OB_SPEC(adresse3,d56&)}})
  fett_pixel%=VAL(CHAR{{OB_SPEC(adresse3,d58&)}})
  unter_pixel%=VAL(CHAR{{OB_SPEC(adresse3,d60&)}})
  hell_maske%=VAL(CHAR{{OB_SPEC(adresse3,d62&)}})
  kursiv_maske%=VAL(CHAR{{OB_SPEC(adresse3,d64&)}})
  mode%=VAL(CHAR{{OB_SPEC(adresse3,d66&)}})
  ARRAYFILL kursiv!(),0
  FOR i%=0 TO 15
    IF (kursiv_maske% AND 2^i%)=0
      kursiv!(i%)=-1
      kursiv!(i%+16)=kursiv!(i%)
      kursiv!(i%+32)=kursiv!(i%)
      kursiv!(i%+48)=kursiv!(i%)
      kursiv!(i%+64)=kursiv!(i%)
    ENDIF
  NEXT i%
  fett_pixel%=VAL(CHAR{{OB_SPEC(adresse3,d58&)}})
  DEFLINE -hell_maske%,0,0,0
  FOR i%=0 TO 72 STEP 2
    LINE i%,200,i%,200+rasterzeilen%
    LINE i%+1,201,i%+1,201+rasterzeilen%
  NEXT i%
  GET 0,201,70,200+rasterzeilen%,hell_m$
```

```
  DEFLINE 1,0,0,0
RETURN
> PROCEDURE zeig_dialogbox(boxadresse)
> PROCEDURE loesch_dialogbox(boxadresse)
```

Zeichensätze auf Diskette zu speichern, ist natürlich eine außerordentlich wichtige Funktion des Programms. Die Kompliziertheit der Zeichensätze macht die folgende Prozedur allerdings recht umfangreich. Es werden zunächst alle vom Programm genutzten und berechneten Werte in den Speicherbereich abgelegt, der den Zeichensatz enthält (Anfangsadresse ist »ad%«). Dann wird der komplette Speicherblock mit »BSAVE« abgespeichert.

```
PROCEDURE zeichensatz_speichern
  PRINT AT(1,1);"ZEICHENSATZ SPEICHERN: ";z_name$
  FILESELECT lwz$+"\*.*","",name$
  IF RIGHT$(name$)="\" OR name$=""
    ALERT 1,"Zeichensatz NICHT|abgespeichert!",1,"WEITER",var
  ELSE
    lwz$=LEFT$(name$,RINSTR(name$,"\")-1)
    poke_word(ad%,f_nr%)
    poke_word(ad%+2,groesse%)
    CHAR{ad%+4}=f_name$
    poke_word(ad%+36,asc_l%)
    poke_word(ad%+38,asc_h%)
    poke_word(ad%+40,tl_bl%)
    poke_word(ad%+42,al_bl%)
    poke_word(ad%+44,hl_bl%)
    poke_word(ad%+46,dl_bl%)
    poke_word(ad%+48,bl_bl%)
    poke_word(ad%+50,b_max%)
    poke_word(ad%+52,b_box%)
    poke_word(ad%+54,l_offset%)
    poke_word(ad%+56,r_offset%)
    poke_word(ad%+58,fett_pixel%)
    poke_word(ad%+60,unter_pixel%)
    poke_word(ad%+62,hell_maske%)
    poke_word(ad%+64,kursiv_maske%)
    poke_word(ad%+66,0) ! mode% auf 0 setzen
    poke_long(ad%+68,hor_tab%)
    poke_long(ad%+72,zei_tab%)
    poke_word(ad%+82,rasterzeilen%)
    breite_gesamt%=0
    z_offs%=0
    z_ad%=ad%+zei_tab%
    poke_word(z_ad%,z_offs%)
```

```
    FOR zei%=asc_l% TO asc_h%
      ADD z_offs%,breite%(zei%)
      ADD breite_gesamt%,breite%(zei%)
      ADD z_ad%,2
      poke_word(z_ad%,z_offs%)
    NEXT zei%
    l_raster%=((breite_gesamt%+15)\16)*2+2
    poke_word(ad%+80,l_raster%)
    ras_tab%=zei_tab%+(asc_h%-asc_l%+2)*2
    poke_long(ad%+76,ras_tab%)
    zeichensatz_uebernehmen
    satz_laenge%=ras_tab%+l_raster%*rasterzeilen%
    BSAVE name$,ad%,satz_laenge%
  ENDIF
RETURN
> PROCEDURE bild_laden
> PROCEDURE grafik_laden(bild_name$,bild_adresse,filenummer)
> PROCEDURE grafikbildschirm
> PROCEDURE matrix_loeschen
> PROCEDURE zeichen_ab !Im Zeichensatz weiterblättern
> PROCEDURE zeichen_uebernehmen
> PROCEDURE zeichen_breiter
> PROCEDURE zeichen_schmaler
```

Die Prozedur »zeichen_in_matrix« übernimmt nicht nur das gerade aktuelle Zeichen in die große Matrix, sondern stellt auch noch die Zeichen in der hellen, kursiven und fetten Schriftart dar.

```
PROCEDURE zeichen_in_matrix
  TEXT 545,40,"   "
  TEXT 545,60,"   "
  TEXT 545,40,STR$(breite%(zeichen%))
  TEXT 545,60,STR$(zeichen%)
  IF breite%(zeichen%)>0
    DEFFILL 1,2,8
    PUT 0,328,zeichen$(zeichen%)          !Normal
   CLR kursiv&
    FOR i%=0 TO rasterzeilen%-1           !Kursiv
      GET 0,328+rasterzeilen%-i%,breite%(zeichen%),
            328+rasterzeilen%-i%,kursiv$
      ADD kursiv&,kursiv!(i%)
      PUT 100-kursiv&,200+rasterzeilen%-i%,kursiv$
    NEXT i%
    FOR i%=0 TO fett_pixel%               !Fett
```

```
      PUT 100+i%,328,zeichen$(zeichen%),7
    NEXT i%
    PUT 0,200,zeichen$(zeichen%)          !Hell
    PUT 0,200,hell_m$,1
    FOR j%=0 TO breite%(zeichen%)-1
      xj1%=320+j%*4
      PUT xj1%,100,raster$
      FOR i%=0 TO rasterzeilen%-1
        yi1%=100+i%*4
        IF PTST(j%,328+i%)
          PBOX xj1%,yi1%,xj1%+4,yi1%+4
        ENDIF
      NEXT i%
    NEXT j%
  ENDIF
RETURN
```

Es folgt die Prozedur, die die Maus innerhalb der großen Matrix kontrolliert. Auch hier wird wieder, wie schon in »menu«, ein »EVNT_MULTI« zur Überwachung der Mausaktivitäten genutzt.

```
PROCEDURE maus_in_matrix
  IF POINT((mx%-320)/4,328+(my%-100)/4)
    DEFFILL 1,0,0
    COLOR 0
  ELSE
    DEFFILL 1,2,8
    COLOR 1
  ENDIF
  REPEAT
    ~EVNT_MULTI(&X100010,1,1,1,0,0,0,0,0,0,0,0,0,0,
             message_adr%,0,mx%,my%,mb%,ks%,ke%,sc%)
    EXIT IF mx%<320 OR mx%>319+breite%(zeichen%)*4
         OR my%<100 OR my%>99+rasterzeilen%*4
    IF mb%=1
      pos_x%=(mx%-320)/4
      pos_y%=(my%-100)/4
      PBOX 320+pos_x%*4,100+pos_y%*4,324+pos_x%*4,104+pos_y%*4
      DRAW pos_x%,328+pos_y%
    ENDIF
  UNTIL mb%=0
  COLOR 1
  DEFFILL 1,2,8
RETURN
```

Die Prozedur »zeichensatz_auswerten« wird aufgerufen, wenn ein neuer Zeichensatz eingeladen wurde. Für die einfachere Handhabung innerhalb unseres Programms gehen wir bei der Auswertung der eigentlichen Zeichenmatrix folgenden Weg: Die Zeichen werden mit der BITBLT-Funktion des GFA-Basic ausgelesen und auf dem Bildschirm dargestellt. Anschließend speichern wir jedes Zeichen mit »GET« in einen String. Dieses Zeichen wird nun evtl. innerhalb des Programms verändert und erst beim Abspeichern des Zeichensatzes zusammen mit allen anderen Zeichen zu einer kompletten Zeichenmatrix zusammengefügt (durch die Prozedur »zeichensatz_uebernehmen«).

```
PROCEDURE zeichensatz_auswerten
  CLS
  PRINT AT(1,13);"Momentchen, bitte ..."
  PRINT AT(1,15);"Ich lese jetzt die Zeichen ein."
  peek_word(ad%,32767,f_nr%)
  peek_word(ad%+2,32767,groesse%)
  f_name$=SPACE$(32)
  LSET f_name$=CHAR{ad%+4}
  peek_word(ad%+36,32767,asc_l%)
  peek_word(ad%+38,255,asc_h%)
  peek_word(ad%+40,99,tl_bl%)
  peek_word(ad%+42,99,al_bl%)
  peek_word(ad%+44,99,hl_bl%)
  peek_word(ad%+46,99,dl_bl%)
  peek_word(ad%+48,99,bl_bl%)
  peek_word(ad%+50,99,b_max%)
  peek_word(ad%+52,99,b_box%)
  peek_word(ad%+54,99,l_offset%)
  peek_word(ad%+56,99,r_offset%)
  peek_word(ad%+58,99,fett_pixel%)
  peek_word(ad%+60,99,unter_pixel%)
  peek_word(ad%+62,32767,hell_maske%)
  peek_word(ad%+64,32767,kursiv_maske%)
  peek_word(ad%+66,15,mode%)
  peek_long(ad%+68,hor_tab%)
  peek_long(ad%+72,zei_tab%)
  peek_long(ad%+76,ras_tab%)
  peek_word(ad%+80,32767,l_raster%)
  peek_word(ad%+82,72,rasterzeilen%)
  peek_long(ad%+84,font_2%)
  bitblit_aus_speicher                      !Parameter für Bitblt
  ARRAYFILL breite%(),0
  FOR z%=1 TO asc_h%-asc_l%
    ascii%=asc_l%+z%-1
    znr%=z%*2-2
```

```
    offs_l%=PEEK(ad%+zei_tab%+znr%)+PEEK(ad%+zei_tab%+znr%+1)*256
    offs_r%=PEEK(ad%+zei_tab%+znr%+2)+PEEK(ad%+zei_tab%+znr%+3)
            *256
    breite%(ascii%)=offs_r%-offs_l%
    IF breite%(ascii%)>0
      par%(0)=offs_l%                     !Linker und rechter Wert
      par%(2)=offs_r%-1                   !für Bitblt
      par%(6)=offs_r%-offs_l%-1
      BITBLT s_mfdb%(),d_mfdb%(),par%()
    ENDIF
    GET 320,0,320+breite%(ascii%)-1,
        rasterzeilen%-1,zeichen$(ascii%)
  NEXT z%
RETURN
PROCEDURE zeichensatz_uebernehmen
  CLS
  PRINT AT(1,13);"Momentchen, bitte ..."
  PRINT AT(1,15);"Ich übernehme und speichere jetzt die Zeichen."
  bitblit_in_speicher
  offs_r%=-1
  FOR ascii%=asc_l% TO asc_h%
    offs_l%=offs_r%+1
    offs_r%=offs_l%+breite%(ascii%)-1
    IF breite%(ascii%)>0
      PUT 320,0,zeichen$(ascii%)
      par%(2)=breite%(ascii%)-1
      par%(4)=offs_l%
      par%(6)=offs_r%
      BITBLT s_mfdb%(),d_mfdb%(),par%()
    ENDIF
  NEXT ascii%
RETURN
```

GFA-Basic bietet bekanntlich mehrere Versionen des BITBLT-Befehls. Wir haben uns in diesem Beispielprogramm für den Befehl mit drei Übergabefeldern entschieden. Diesen Feldern müssen jeweils die richtigen Werte (Adressen und Koordinaten) übergeben werden, was zum Teil in den zwei folgenden Prozeduren geschieht. Die restlichen Parameter werden in den Prozeduren »zeichensatz_uebernehmen« und »zeichensatz_auswerten« definiert.

```
PROCEDURE bitblit_aus_speicher          !Werte für Bitblt
  s_mfdb%(0)=ad%+ras_tab%
  s_mfdb%(1)=l_raster%*8
  s_mfdb%(2)=rasterzeilen%
  s_mfdb%(3)=s_mfdb%(1)/16
  s_mfdb%(4)=0
  s_mfdb%(5)=1
  d_mfdb%(0)=XBIOS(2)+40
  d_mfdb%(1)=640
  d_mfdb%(2)=rasterzeilen%
  d_mfdb%(3)=d_mfdb%(1)/16
  d_mfdb%(4)=0
  d_mfdb%(5)=1
  par%(1)=0
  par%(3)=rasterzeilen%-1
  par%(4)=0
  par%(5)=0
  par%(7)=rasterzeilen%-1
  par%(8)=3
RETURN
PROCEDURE bitblit_in_speicher           !Werte für Bitblt
  s_mfdb%(0)=XBIOS(2)+40
  s_mfdb%(1)=640
  s_mfdb%(2)=rasterzeilen%
  s_mfdb%(3)=s_mfdb%(1)/16
  s_mfdb%(4)=0
  s_mfdb%(5)=1
  d_mfdb%(0)=ad%+ras_tab%
  d_mfdb%(1)=l_raster%*8
  d_mfdb%(2)=rasterzeilen%
  d_mfdb%(3)=d_mfdb%(1)/16
  d_mfdb%(4)=0
  d_mfdb%(5)=1
  par%(0)=0
  par%(1)=0
  par%(3)=rasterzeilen%-1
  par%(4)=0
  par%(5)=0
  par%(7)=rasterzeilen%-1
  par%(8)=3
RETURN
> PROCEDURE ende
> PROCEDURE peek_word(adresse%,maximal%,VAR wert%)
```

```
> PROCEDURE peek_long(adresse%,VAR wert%)
> PROCEDURE poke_word(adresse%,wert%)
> PROCEDURE poke_long(adresse%,wert%)
```

Durch unseren Zeichensatz-Editor werden Sie sicher auch erkannt haben, daß zur Auswertung und Darstellung eines Zeichensatzes statt GDOS auch ein GFA-Basic-Programm genutzt werden kann. Durch die BITBLT-Routine können die Zeichen aus dem Zeichensatzraster auf den Bildschirm kopiert werden. Natürlich gehen dabei alle besonderen Möglichkeiten, die Ihnen das GEM bietet (Textattribute), verloren bzw. müssen wie in unserem Programm umständlich nachgebildet werden.

Nach dem Start des Programms 7_1.GFA müssen Sie sich entscheiden, ob Sie einen völlig neuen Zeichensatz entwickeln, oder einen schon bestehenden ändern möchten. Soll es ein neuer Zeichensatz sein, können Sie in der folgenden Dialogbox angeben, wie viele Rasterzeilen Sie benötigen. Diese Einstellung kann nur einmal zu Beginn des Programms gemacht werden. Die übrigen Einstellungen können dagegen jederzeit geändert werden. Sie sehen, daß einige Felder der Dialogbox hell gezeichnet sind, also nicht angewählt werden können. Es handelt sich dabei um die Parameter, die von unserem Programm automatisch ermittelt werden. Wird ein vorhandener Zeichensatz eingeladen, dienen diese Angaben als Information für Sie. Wird das Feld »Weiter« angeklickt oder die [Return]-Taste gedrückt, erscheint der Eingabebildschirm. Hier kann nun für jedes Zeichen eine beliebige Zeichenbreite eingestellt werden. Dies geschieht durch Anklicken der reversen Pfeile im oberen rechten Bildschirmbereich.

Im Pull-down-Menü »Optionen« können Sie die eingangs besprochene Dialogbox aufrufen. Hier können Sie auch eine Grafik einladen, um aus dieser heraus Ausschnitte für eigene Zeichen zu wählen (erreichen Sie durch das Feld »Grafikbildschirm«).

Ansonsten ist das Programm einfach zu bedienen. Spielen Sie einfach ein wenig damit herum, um die Funktionen kennenzulernen.

Kommen wir zurück zum GDOS. Der einfachste Weg, neue Zeichensätze zu nutzen, ist also, das GDOS zu nutzen. Sie müssen das Programm dazu auf Ihrer Boot-Diskette im AUTO-Ordner abspeichern. Es wird dann nach dem Einschalten des Systems automatisch mitgeladen. Neue Zeichensätze können Sie dann allerdings noch nicht nutzen. Diese müssen nämlich durch die Verwendung einer VDI-Funktion nachgeladen werden. Unser GFA-Basic liefert Ihnen diese Routine, allerdings nur, wenn das GDOS auch wirklich beim Systemstart mitgebootet wurde. Denn nur durch GDOS ist das VDI Ihres Atari ST in der Lage, Gerätetreiber und Zeichensätze einzuladen.

In Ihrem GFA-Basic-Programm sollte zunächst also überprüft werden, ob das GDOS überhaupt vorhanden ist. Dies geschieht einfach durch die Abfrage

```
a=GDOS?
```

Sie erhalten als Ergebnis –1, also den Wert für TRUE, wenn GDOS verfügbar ist, bzw. 0 (FALSE), wenn das Programm nicht mitgebootet wurde.

Den weiteren Verlauf Ihres Programms sollten Sie also von dieser Abfrage abhängig machen.

Es steht Ihnen jetzt eine Funktion des GFA-Basic zur Verfügung, die die Gerätetreiber und Zeichensätze nachlädt, die in der ASCII-Datei ASSIGN.SYS Ihrer Boot-Diskette angegeben werden. Diese Funktion nutzt eine VDI-Funktion gleichen Namens. Sie lädt so viele Zeichensätze und Treiber, wie in der Datei angegeben werden und maximal in Ihren Speicher passen.

anzahl=VST_LOAD_FONTS(0)

Es ist hier als Parameter »0« vorgesehen. Erst bei möglichen Erweiterungen des VDI könnten andere Werte einen Sinn bekommen. In »anzahl« wird Ihnen die Anzahl der zusätzlichen Zeichensätze zurückgeliefert.

fontnummer=VQT_NAME(laufende_nummer,fontname$)

Hier wird in »laufende_nummer« angegeben, von welchem Zeichensatz Sie Informationen wünschen. Die Nummer kann Werte von 1 bis »anzahl+1« annehmen. Mit der ermittelten Fontnummer können Sie nun beim Befehl DEFTEXT den Zeichensatz einschalten und Text in der gewünschten Schriftart ausgeben. Zusätzlich erhalten Sie in »fontname$« den Namen des Zeichensatzes zurück.

Wenn Ihnen die zusätzlich benötigten Zeichensätze bekannt sind, brauchen Sie die vorstehende Funktion nicht aufzurufen. Ihr Zweck ist es nämlich, die Fontnummer eines Zeichensatzes zu ermitteln. Wenn Sie diese Nummer vorher schon kennen, können Sie sie im weiteren Verlauf Ihres Programms verwenden. Allerdings kann man vorher nicht wissen, ob der gewünschte Zeichensatz überhaupt geladen werden konnte. Möglicherweise reichte ja der reservierte Speicherplatz nicht aus.

Wenn nur ein bestimmter Zeichensatz in Ihrem Programm benötigt wird, empfiehlt es sich, in der Datei ASSIGN.SYS auch nur diesen Zeichensatz anzugeben. Bedenken Sie dabei, daß die Zeichensätze in der Regel aus mehreren Dateien bestehen, um mehr Größen zu ermöglichen. Natürlich müssen dann alle Dateinamen in der ASCII-Datei angegeben werden. Übrigens wird ASSIGN.SYS schon beim Booten eingelesen. Sie können also danach keine Änderungen mehr an den Daten vornehmen, ohne anschließend den Rechner neu zu starten oder die Reset-Taste zu drücken.

Zum Experimentieren haben wir Ihnen auf der beiliegenden Diskette einige Zeichensatzdateien mitgeliefert. Die Dateien tragen die Namen FUTUR10.FNT, ALT10.FNT und NEU10.FNT, wie auch die Zeichensätze selbst. Die Fontnummern sind 3000, 3001 und 3002. Ebenfalls enthalten ist eine Datei ASSIGN.SYS, die die Namen unseres Zeichensatzes enthält.

7.2 Der Befehl RC_COPY

Die vielfältigen Befehle zum Kopieren und Verschieben rechteckiger Bildschirmbereiche wurden in diesem Buch schon mehrmals angesprochen. Einer dieser Befehle ist »RC_COPY«. Er ist etwas einfacher anzuwenden als die BITBLT-Befehle, da er mit wesentlich weniger Parametern auskommt. Die Programmierung ist sogar recht einfach, wie Sie noch sehen werden.

RC_COPY s_adr,x,y,breite,hoehe TO d_adr,x1,y1[,modus]

Die Bedeutung der Parameter:

s_adr	Anfangsadresse des Quell-Bildschirmspeichers, aus dem kopiert werden soll (source address).
x	Linke X-Koordinate des Rechtecks im Ursprungsbild.
y	Obere Y-Koordinate des Rechtecks im Ursprungsbild.
breite	Breite des Rechtecks.
hoehe	Höhe des Rechtecks.
d_adr	Anfangsadresse des Ziel-Bildschirmspeichers (destination address).
x1	Linke X-Koordinate, an die das Rechteck kopiert werden soll.
y1	Obere Y-Koordinate, an die das Rechteck kopiert werden soll.
modus	Verknüpfungsmodus zwischen Quell- und Zielbereich (Werte zwischen 0 und 15, Bedeutung wie bei BITBLT und PUT).

Es müssen hier unter anderem die Anfangsadressen des Quell-Bildschirms und des Ziel-Bildschirms angegeben werden. Somit ist also auch mit »RC_COPY« das Kopieren zwischen zwei Bildschirmspeichern möglich, allerdings sehr viel einfacher als bei »BITBLT«.

Als Anwendungsbeispiel für diesen Befehl haben wir das folgende Programm geschrieben. Wir nutzen »RC_COPY« hier dazu, einen kompletten Bildschirminhalt in einen anderen Bildschirmspeicher zu kopieren, allerdings nicht »in einem Stück«, sondern zeilenweise mit jeweils nur einem Pixel Breite. Sie können so sehr nette Überblendeffekte zwischen zwei Bildern erhalten. In Ihren Programmen kann das zum Beispiel dazu genutzt werden, von einem Titelbild in den Arbeitsbildschirm überzublenden.

Im folgenden Programmbeispiel haben wir einmal fünf verschiedene Möglichkeiten für das Vorhergesagte programmiert. Sie finden das Programm auf der beiliegenden Diskette im Ordner 7.GFA als Programm 7_2.GFA.

Nach dem Programmstart werden zwei zusätzliche Bildschirmspeicher reserviert, in die einige Grafiksymbole gezeichnet werden. Zum Umschalten in diese Bildschirme nutzen wir XBIOS(5...). Mit dieser Funktion können bekanntlich die Anfangsadressen des logischen und des physikalischen Bildschirmspeichers eingestellt werden. In unserem Beispiel bleibt der physikalische Bildschirmspeicher allerdings unverändert, so daß auf dem Bildschirm auch nicht sichtbar ist, was in die neuen Bildschirmspeicher gezeichnet wird.

Im Anschluß werden logischer und physikalischer Bildschirm wieder auf dieselbe Adresse gesetzt und nacheinander die Prozeduren zum Kopieren aufgerufen. Das Programm kann nur durch [Control]+[Shift]+[Alternate] oder durch anhaltendes Drücken der rechten Maustaste abgebrochen werden.

Wir möchten das Programm hier nicht vollständig abdrucken, sondern beschränken das Listing auf zwei Routinen, in denen der »RC_COPY«-Befehl genutzt wird.

```
PROCEDURE copy1(source,destination,modus)
  LOCAL i%
  FOR i%=1 TO 200
    RC_COPY source,0,200-i%,640,1 TO destination,0,200-i%,modus
    RC_COPY source,0,199+i%,640,1 TO destination,0,199+i%,modus
  NEXT i%
RETURN
PROCEDURE copy4(source1,source2,destination,modus)
  LOCAL i%,j%,s%
  s%=640/400
  FOR k%=1 TO 2
    CLR j%
    FOR i%=0 TO 399
      RC_COPY source1,i%,i%,640-i%,1 TO destination,i%,i%,modus
      RC_COPY source1,j%,0,1,400 TO destination,j%,0,modus
      ADD j%,s%
    NEXT i%
    FOR i%=399 DOWNTO 0
      RC_COPY source2,j%,i%,640-j%,1 TO destination,j%,i%,modus
      RC_COPY source2,j%,i%,1,400-i% TO destination,j%,i%,modus
      SUB j%,s%
    NEXT i%
  NEXT k%
RETURN
```

Wir kopieren jeweils nur einzelne Pixel-Zeilen bzw. -Spalten. Den Prozeduren werden dazu lediglich die Anfangsadressen der Bildschirmspeicher sowie der Verknüpfungsmodus übergeben.

Weiter geht es jetzt mit der Maschinensprache und deren Aufruf aus dem Basic heraus.

7.3 Aufruf von Maschinenprogrammen

Mit dem GFA-Basic lassen sich schon recht schnell ablaufende Programme schreiben. In einigen Fällen reicht die Geschwindigkeit jedoch nicht aus. Der bequemste Weg, das Basic-Programm zu beschleunigen, besteht darin, es zu kompilieren. Das bringt in den meisten Fällen schon die geforderte Geschwindigkeit, reicht manchmal aber dennoch nicht aus. Die Programmierer haben auch für diesen sicherlich nicht sehr oft eintretenden Fall vorgesorgt. Es gibt Befehle, die das Aufrufen von C-Programmen oder Maschinenprogrammen aus dem Basic heraus unterstützen. In diesem Abschnitt befassen wir uns mit dem Aufruf von Maschinenprogrammen.

Es bietet sich folgende Vorgehensweise bei der Erstellung des Programms an:

Zuerst schreibt man das Basic-Programm, von dem später die Maschinensprachroutine aufgerufen werden soll. Danach schreibt man mit Hilfe eines Assemblers das Maschinenprogramm, es muß mit ».RTS« enden, und assembliert es. Das dadurch entstehende Programm wandelt man am besten mit Hilfe eines Hilfsprogramms in Data-Zeilen um. Wenn Sie ein solches Programm schreiben, denken Sie bitte daran, daß die ersten 28 Byte des Maschinenprogramms nicht benötigt werden. Das eigentliche Programm beginnt also erst ab Byte 29. Den Schluß des Programms bilden, je nach verwendetem Assembler, 4 oder auch 8 Nullbyte. Die Datei, die die Datazeilen enthält, merged man zum Basic-Programm. Mittels einer Schleife poked das Programm die Daten in einen vorher reservierten Speicherbereich des Rechners. Wir beschreiben jetzt den Aufruf des Programms.

In den meisten Fällen sollen einem Maschinenprogramm Daten übergeben werden, die es dann weiterverarbeitet. Das oder die Ergebnisse werden zum Schluß an das Basic-Programm zurückgegeben. GFA 3.0 bietet uns dazu einen besonders komfortablen Befehl an. Es gibt auch noch einen zweiten, auf den wir aber nicht weiter eingehen:

RCALL Adr%,Prozessor_register%()

Die Variable »Adr%« enthält die Anfangsadresse des Maschinenprogramms. Der nächste Parameter stellt den Namen eines Long-Integer-Feldes dar, das für mindestens 16 Feldeinträge dimensioniert wurde. Die Parameterübergabe zum Maschinenprogramm und von ihm zurück zum Basic-Programm erfolgt über die Prozessorregister. Jedem Eintrag des Feldes ist dabei ein Register des 68000 zugeordnet. Die Zuordnung:

Feldindex	entspricht dem Register...
0	d0
1	d1
2	d2
3	d3

Feldindex	entspricht dem Register...
4	d4
5	d5
6	d6
7	d7
8	a0
9	a1
10	a2
11	a3
12	a4
13	a5
14	a6
15	a7 nur Rückgabe

Vor dem Aufruf belegt man das Feld, seinen Erfordernissen entsprechend, vor. GFA kopiert nach »RCALL« den Inhalt des gesamten Feldes in die entsprechenden Register. Ist das Maschinenprogramm fertig, kopiert GFA, vor der Rückkehr ins Basic, den Inhalt der Register in das Feld. Der Aufruf des Maschinenprogramms erfolgt im User-Modus des Prozessors.

Es ist wohl das beste, wenn wir Ihnen das Ganze einmal anhand eines Demonstrationsprogramms zeigen. Bei unserer Arbeit an diesem Buch, speziell im Grafikkapitel, benötigten wir eine Routine, die beliebige Bildschirmbereiche spiegelt. Nachdem sie in Basic programmiert war, nervte uns ihre lange Ausführungszeit gar sehr. Optimierung brachte auch nicht den gewünschten Erfolg, man bedenke, daß beim vertikalen Spiegeln eines ganzen Bildschirms immerhin 256.000 Pixel bewegt werden müssen. Es reifte der Entschluß, diese Arbeit einem Maschinenprogramm zu übertragen. Jetzt geht auch beim Spiegeln »die Post ab«. Vertikales Spiegeln eines ganzen Bildschirms dauert nur noch ca. eine halbe Sekunde (das ist etwa 14mal schneller als das Basic-Programm), horizontales Spiegeln ca. $^{6}/_{100}$ Sekunden (das ist ca. 11mal schneller). Hinzu kommt, daß sich das Maschinenprogramm bestimmt noch optimieren läßt. Uns ist es aber in der vorliegenden Version schnell genug.

Sie finden eine Version des Programms unter dem Namen 7_3.GFA auf der beiliegenden Diskette. Das Programm baut einen Bildschirm auf, mit der Maus läßt sich ein Bereich bestimmen, der horizontal oder vertikal gespiegelt werden kann. Es läßt sich sehr leicht folgender Bildaufbau erreichen:

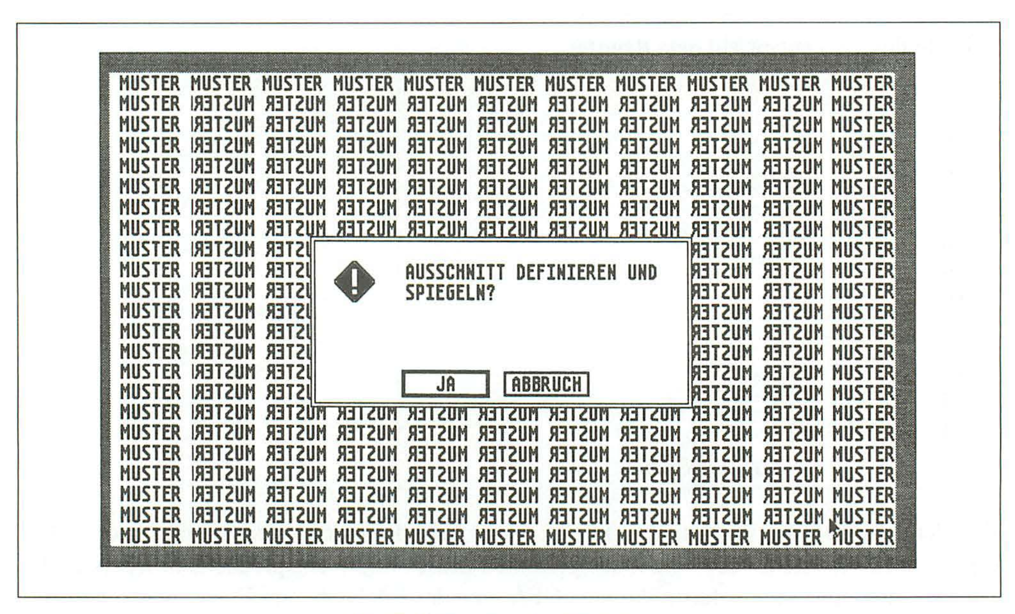


Bild 7.3: Gespiegelter Bildschirm

Zum Programm:

```
' ************************** BEMU 89 **************************
' ********** AUFRUF VON MASCHINENPROGRAMMEN (DEMO 1) *********
' ************** Programm aus dem GfA 3.0 Buch ***************
' ************ Von Wilhelm Besenthal & Jens Muus *************
' ********* erschienen im Markt & Technik Verlag 1989 ********
' ************************************************************
'
ON MENU BUTTON 1,1,1 GOSUB ausschnitt
programm$=SPACE$(238)                  !Platz für Maschinenprogramm schaffen
adprg%=VARPTR(programm$)               !Anfangsadresse des Maschinenprogramms
DIM r%(16)                             !Feld für Prozessorregister
```

Das Maschinenprogramm hat eine Länge von 238 Byte. Wir reservieren den Platz in der Variablen »programm$«. Die Anfangsadresse entspricht dem ersten Byte des Strings, wir erfragen sie mit dem Befehl »VARPTR«. Als nächsten Schritt richtet das Programm das Feld für die Prozessor-Register ein.

```
'
DO                                     !Schleife poked M.Prg in den Speicher
  READ a%
  EXIT IF a%=-1
```

```
  DPOKE adprg%+i%,a%
  ADD i%,2
LOOP
```

Die Schleife liest die Datas in die Variable »a%« und poked sie in den reservierten Speicher. Nach Beendigung der Schleife steht das Programm im Speicher.

```
'
PRINT
FOR i&=1 TO 22
  PRINT STRING$(11,"MUSTER ")                 !Bildschirm aufbauen
NEXT i&
'
DO
  ALERT 1," AUSSCHNITT DEFINIEREN UND| SPIEGELN?",1," JA |ABBRUCH",bu%
  IF bu%=1
    ALERT 1," VERTIKAL ODER | HORIZONTAL?",1," VERT | HORI ",modus%
    DEC modus%
    GOSUB abfrage
  ELSE
    END
  ENDIF
```

Die Zeilen dienen dem Bildschirmaufbau und der Festlegung des Spiegel-Modus. Nach der Festlegung des zu spiegelnden Bildschirmausschnitts, erfolgt in der Prozedur »abfrage«, kopiert das Programm ihn in die Stringvariable »bild$«.

```
GET x%,y%,x%+b%-1,y%+h%-1,bild$             !Ausschnitt ausschneiden
```

Den nächsten Abschnitt lesen Sie bitte besonders gut durch. Er enthält wichtige Hinweise auf den Einsatz der Maschinenroutine in anderen von Ihnen selbst erstellten Programmen.

```
'
' WICHTIGE BERECHNUNGEN
'
wort_anz%=((LEN(bild$)-6)/h%)/2            !Worte pro Zeile
bit_z%=wort_anz%*16                        !Bits pro Zeile
bit_rest%=bit_z%-(DPEEK(VARPTR(bild$))+1) !Bitrest berechnen
```

Die erste Zeile berechnet die Anzahl der Wörter (ein Wort entspricht zwei Byte) pro Zeile. Der nächste Schritt ist die Berechnung eines evtl. vorhandenen Bit-Restes im letzten Wort einer Zeile. Zu Erinnerung: Eine Bildschirmzeile eines mit »GET« eingelesenen Bildschirmausschnitts belegt in der Stringvariablen immer ein oder mehrere Wörter. Das letzte Wort ist entweder vollständig oder nur teilweise mit Pixeln belegt, das hängt von der Breite des Ausschnitts ab. Die Anzahl der Bits des letzten Wortes, die nichts mehr mit dem Ausschnitt zu tun haben, steht nach der Berechnung in der Variablen »bit_rest%«.

```
'
' BELEGUNG DER PROZESSOR-REGISTER VORBEREITEN
'
r%(0)=wort_anz%                    !Wortanzahl einer Zeile
r%(1)=h%                           !Höhe des Ausschnitts
r%(2)=bit_rest%                    !Bitrest, der nicht gespiegelt wird
r%(3)=modus%                       !Modus
r%(8)=VARPTR(bild$)+6              !Quell-Adresse
r%(9)=VARPTR(bild$)+6              !Ziel-Adresse
```

Dem Maschinenprogramm müssen folgende Daten übergeben werden: Anzahl der Wörter einer Zeile, die Höhe des zu spiegelnden Ausschnitts, der schon besprochene Bit-Rest (nur beim vertikalen Spiegeln), der Modus (0 = vertikal spiegeln, 1 = horizontal spiegeln), die Adresse des Speichers, ab der der mit den anderen Parametern beschriebene Ausschnitt zu finden ist, und die Adresse, ab der der gespiegelte Ausschnitt nach dem Spiegeln stehen soll.

Bei unserem Beispielprogramm ist die Quell-Adresse gleich der Ziel-Adresse. Soll der gespiegelte Ausschnitt später in einer anderen Stringvariablen zu finden sein, fügen Sie einfach nach dem Einlesen mit »GET« folgende Zeile ein:

```
Bild1$=Bild$
```

Der Feldeintrag »r%(9)« heißt dann entsprechend:

```
r%(9)=VARPTR(bild1$)+6             !Ziel-Adresse
```

Die Vorarbeiten sind getan, wir kommen zum Aufruf des Maschinenprogramms.

```
  '
  adprg%=VARPTR(programm$)         !Anfangsadresse des Maschinenprogramms
  RCALL adprg%,r%()                !Maschinenprogramm aufrufen
```

Die Anfangsadresse des Programms haben wir zwar schon oben erfragt. Hat das Basic aber in der Zwischenzeit eine Garbage Collection (Entfernen aller überflüssigen Variablen aus dem Speicher) vorgenommen, dann stimmt sie höchstwahrscheinlich nicht mehr. Wir erfragen sie darum nochmals. Der nächste Befehl ruft dann endlich das Maschinenprogramm auf. Vor dem eigentlichen Aufruf kopiert es die Werte des Feldes »r%()« in die entsprechenden Prozessor-Register.

```
  PUT x%,y%,bild$                  !Spiegelbild darstellen
LOOP
```

Nach der Rückkehr aus dem Maschinenprogramm bringt der PUT-Befehl den gespiegelten Ausschnitt auf den Bildschirm. Programmierer, die eine andere Zieladresse verwandten, müssen ihn entsprechend ändern, Beispiel:

```
PUT x%,y%,bild1$.
```

Die Prozeduren, die das Auswählen des Bildschirmausschnitts ermöglichen, beschreiben wir im Schnelldurchlauf.

```
'
PROCEDURE abfrage
  b%=0
  REPEAT
    ON MENU
  UNTIL b%>0
RETURN
PROCEDURE ausschnitt
  x%=MOUSEX
  y%=MOUSEY
  ~GRAF_RUBBERBOX(x%,y%,0,0,b%,h%)
RETURN
```

In der Prozedur »ausschnitt« befindet sich ein AES-Aufruf, den es noch zu beschreiben gilt. Sein Name:

~GRAF_RUBBERBOX(P1,P2,P3,P4,R1,R2)

Der Aufruf zeichnet bei gedrückter linker Maustaste ein Rechteck mit veränderbarer Größe (durch Mausbewegung) auf den Bildschirm. Die Koordinaten der linken oberen Ecke übergibt man in P1 (X-Position) und P2 (Y-Position), die Anfangsgröße in P3 (Breite) und P4 (Höhe). Nach dem Loslassen der linken Maustaste stehen in R1 und R2 Höhe und Breite des End-Rechtecks.

Hinweis: Die Koordinaten für X und Y dürfen nicht in der »Menüleiste« liegen. Die Funktion verweigert sonst ihren Dienst.

Den Schluß des Programms bilden die Datazeilen. In ihnen steht, wie schon beschrieben, das Maschinenprogramm.

```
'
DATA 45692,0,28434,45180,0,28428,46716,0
DATA 26446,46716,1,26370,20085,45692,1,28664
DATA 14849,15360,58182,51910,46024,26112,30,54213
DATA 37830,58182,57929,14336,16144,12497,13023,21316
DATA 28406,37830,21313,26350,20085,54213,37830,58182
DATA 14336,13016,21316,28410,37830,21313,26354,20085
DATA 15420,0,46204,0,26374,15420,16,40002
DATA 21312,21313,13824,14360,57940,58197,57940,58197
DATA 57940,58197,57940,58197,57940,58197,57940,58197
DATA 57940,58197,57940,58197,57940,58197,57940,58197
DATA 57940,58197,57940,58197,57940,58197,57940,58197
```

```
DATA 57940,58197,57940,58197,16133,20939,65466,13824
DATA 3138,0,26368,24,14367,58732,14871,60525
DATA 55365,12996,20939,65522,20937,65434,20085,13023
DATA 20939,65532,20937,65422,20085,0,0
DATA -1
```

Das Ihnen eben vorgestellte Programm vergeudet Speicherplatz und ist der Übersicht nicht sehr zuträglich. Die Datas, die das Maschinenprogramm beschreiben, sind beim Programmlauf doppelt vorhanden. Einmal als Data-Zeilen und ein weiteres Mal im Speicher, nach dem Poken. Mit GFA geht's auch anders. Auf der Diskette finden Sie eine weitere Version des oben vorgestellten Programms. Es hat den Namen 7_4.GFA.

Das Programm weist folgende Besonderheiten auf: Kein umständliches Poken des Maschinenprogramms in den Speicher, Spiegeln eines beliebigen Speicherausschnitts über die BITBLT-Funktion. Der zu spiegelnde Speicherbereich muß allerdings »ausgeschnitten« werden, da das Maschinenprogramm keinen Offset zum ersten Wort bzw. ersten Bit des zu spiegelnden Bereichs zuläßt, eine Einschränkung mit der sich unserer Meinung nach leben läßt.

Das Listing, allerdings verkürzt:

```
' ************************* BEMU 89 **************************
' ********** AUFRUF VON MASCHINENPROGRAMMEN (DEMO 2) *********
' ************** Programm aus dem GfA 3.0 Buch ***************
' ************ Von Wilhelm Besenthal & Jens Muus *************
' ********* erschienen im Markt & Technik Verlag 1989 ********
' ************************************************************
'
ON MENU BUTTON 1,1,1 GOSUB ausschnitt
DIM r%(16)                              !Feld für Prozessorregister
DIM bb%(30)
```

Schon der Programmanfang unterscheidet sich von der zuerst vorgestellten Version. Das Programm reserviert keinen Speicher mehr für das Maschinenprogramm. Das Feld »bb%()« wird für die BITBLT-Funktion dimensioniert. Wer von Ihnen diese Funktion noch nicht kennt, der lese bitte im Kapitel 4 über die Fensterverwaltung nach, wir haben sie dort ausführlich beschrieben.

Die nächste Funktion hat es in sich.

```
' DER FOLGENDE AUFRUF ENTHÄLT DAS MASCHINENPROGRAMM
INLINE adprg%,238
```

Mit der INLINE-Funktion können Sie Speicher im Basic-Programm reservieren oder auch mit Daten belegen. Der Speicherbereich, seine Größe gibt der zweite Parameter an (in unserem Fall 238 Byte), wird mit dem Basic-Programm geladen und gespeichert. Die Übertragung von Festdaten in den Speicher ist darum nur einmal erforderlich. Die Variable

»adprg%« enthält nach dem Funktionsaufruf die Anfangsadresse des reservierten Speicherbereichs. Der Funktion dürfen keine Kommentare folgen.

Im vorliegenden Programm befindet sich die Maschinensprache-Routine hinter der INLINE-Funktion. Wir haben das Programm folgendermaßen dorthin bekommen:

Wir änderten die erste Version des Programms dahingehend, daß sie nach dem Poken des Maschinenprogramms in die Stringvariable »programm$«, deren Inhalt mittels BSAVE in eine Datei des Diskettenlaufwerks schrieb. Der nächste Schritt war das Schreiben von: »INLINE adprg%,238« an die entsprechende Programmstelle. Nach dem Schreiben brachte das Drücken der [Help]-Taste ein Untermenü in der Menüzeile des GFA-Editors zum Vorschein. Über Load wurde die mit »BSAVE« erzeugte Datei geladen. Das war alles. Seit dem ersten Speichern des gesamten Basic-Programms wird das Maschinenprogramm beim Laden des Basic-Programms ebenfalls mitgeladen, seine Anfangsadresse ist immer in der Variablen »adprg%« zu finden (die Adresse wird selbstverständlich neueren Speicheraufteilungen angepaßt).

Zurück zum Programm. Den Aufbau des Bildschirms usw. kennen Sie schon, wir machen weiter mit der Belegung der BITBLT-Arrays.

```
  wort_anz%=INT((b%+15)/16)          !WORTE PRO ZEILE
  bit_z%=wort_anz%*16                !BITs PRO ZEILE
  bit_rest%=bit_z%-b%                !BITREST BERECHNEN
  bild$=SPACE$(wort_anz%*2*h%)       !SPEICHERPLATZ FÜR AUSSCHNITT
```

Zunächst berechnet das Programm wieder die Anzahl der Wörter pro Zeile und den Bit-Rest. Die Stringvariable »bild$« reserviert Speicher für die BITBLT-Funktion. Sollen mehr als 32.767 Byte gespiegelt werden, müssen Sie den erforderlichen Speicher über »RESEVE« und »MALLOC« reservieren.

Nach den Berechnungen belegt das Programm die entsprechenden Feldeinträge mit den errechneten und bekannten Werten.

```
  ' BELEGUNG DER PROZESSOR-REGISTER VORBEREITEN
  '
  r%(0)=wort_anz%                    !Wortanzahl einer Zeile
  r%(1)=h%                           !Höhe des Ausschnitts
  r%(2)=bit_rest%                    !Bitrest, der nicht gespiegelt wird
  r%(3)=modus%                       !Modus
  r%(8)=VARPTR(bild$)                !Quell-Adresse
  r%(9)=VARPTR(bild$)                !Ziel-Adresse
  '
  GOSUB ausschneiden(x%,y%,b%,h%)
  RCALL adprg%,r%()                  !Maschinenprg aufrufen
  GOSUB bild_zeigen(x%,y%,b%,h%)
LOOP
```

Es folgt die Beschreibung der Unterprogramme, die das Ausschneiden eines zu spiegelnden Speicherbereichs und sein Restaurieren möglich machen.

Zunächst das »Ausschneiden«:

```
> PROCEDURE ausschneiden(x%,y%,b%,h%)
  HIDEM                            !Maus aus
  bb%(0)=b%                        !Breite des Rasters der Quelle
  bb%(1)=h%                        !Höhe des Rasters der Quelle
  bb%(2)=1                         !Anzahl der Farbebenen
  bb%(3)=1                         !Vordergrundfarbe
  bb%(4)=0                         !Hintergrundfarbe
  bb%(5)=&H3030303                 !Verknüpfungsmodus (Ziel=Quelle)
  bb%(6)=x%                        !X-Koordinate Quelle
  bb%(7)=y%                        !Y-Koordinate Quelle
  bb%(8)=XBIOS(2)                  !Adresse der Quelle
```

Unser Demo bezieht sich auf den aktuellen Bildschirm. Seine Anfangsadresse bekommen wir über die XBIOS-Funktion.

```
  bb%(9)=2                         !Offset zum nächsten Word (von) in Byte
  bb%(10)=80                       !Offset zur nächsten Zeile (von) in Byte
```

Jede Zeile des Bildschirms belegt 80 Byte.

```
  bb%(11)=2                        !Offset zur nächsten Farb-Ebene Quelle)
  bb%(12)=0                        !X-Koordinate Ziel
  bb%(13)=0                        !Y-Koordinate Ziel
  bb%(14)=VARPTR(bild$)            !Adresse des Ziels
  bb%(15)=2                        !Offset zum nächsten Word (Ziel)
  bb%(16)=wort_anz%*2              !Offset zur nächsten Zeile (Ziel)
```

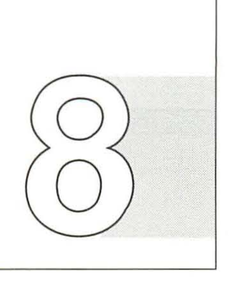
Für das Ziel muß noch die Byte-Anzahl pro Zeile errechnet werden.

```
  bb%(17)=2                        !Offset zur nächsten Farb-Ebene (Ziel)
  bb%(18)=0                        !Zeiger auf Tabelle mit Füllmuster
  bb%(19)=0                        !Offset zur nächsten Zeile der Füllmaske
  bb%(20)=0                        !Offset zur nächsten Farbe der Füllmaske
  bb%(21)=0                        !Höhe der Füllmaske
  BITBLT bb%()                     !Bit-Block verschieben
  SHOWM                            !Maus wieder ein
RETURN
```

Sie können selbstverständlich den Ausschnitt vor dem Spiegeln mit einer beliebigen Füllmaske verknüpfen. Wie das geht, haben wir bei der Fensterprogrammierung beschrieben. Die folgende Prozedur bringt den gespiegelten Ausschnitt wieder auf den Bildschirm. Da sie ähnlich aufgebaut ist, enthalten wir uns jeglichen Kommentars.

```
PROCEDURE bild_zeigen(x%,y%,b%,h%)
  HIDEM                              !Maus aus
  bb%(0)=b%                          !Breite des Rasters der Quelle
  bb%(1)=h%                          !Höhe des Rasters der Quelle
  bb%(2)=1                           !Anzahl der Farb-Ebenen
  bb%(3)=1                           !Vordergrundfarbe
  bb%(4)=0                           !Hintergrundfarbe
  bb%(5)=&H3030303                   !Verknüpfungsmodus (Ziel=Quelle)
  bb%(6)=0                           !X-Koordinate Quelle
  bb%(7)=0                           !Y-Koordinate Quelle
  bb%(8)=VARPTR(bild$)               !Adresse der Quelle
  bb%(9)=2                           !Offset zum nächsten Word (von) in Byte
  bb%(10)=wort_anz%*2                !Offset zur nächsten Zeile (von) in Byte
  bb%(11)=2                          !Offset zur nächsten Farb-Ebene (Quelle)
  bb%(12)=x%                         !X-Koordinate Ziel
  bb%(13)=y%                         !Y-Koordinate Ziel
  bb%(14)=XBIOS(2)                   !Adresse des Ziels
  bb%(15)=2                          !Offset zum nächsten Word (Ziel)
  bb%(16)=80                         !Offset zur nächsten Zeile (Ziel)
  bb%(17)=2                          !Offset zur nächstn Farb-Ebene (Ziel)
  bb%(18)=0                          !Zeiger auf Tabelle mit Füllmuster
  bb%(19)=0                          !Offset zur nächsten Zeile der Füllmaske
  bb%(20)=0                          !Offset zu nächsten Farbe der Füllmaske
  bb%(21)=0                          !Höhe der Füllmaske
  BITBLT bb%()                       !Bit-Block verschieben
  SHOWM                              !Maus wieder ein
RETURN
```

Sie wissen jetzt, wie man mit einfachen Mitteln ein Maschinenprogramm von Basic aus aufruft. Wenn Sie dabei wie oben beschrieben vorgehen, dürfte beim Verwirklichen eigener Projekte in Maschinensprache eigentlich nichts mehr passieren.



Das Resource Construction Set

Dem GFA-Basic 3.0 ist ein nützliches Hilfsprogramm beigefügt, das RCS2.PRG, ein *Resource Construction Set*. Das Programm dient dazu, auf einfache Weise Dialogboxen, Menüleisten usw. zu erstellen. Wir gehen davon aus, daß Sie die Grundbegriffe für die Arbeit mit GEM kennen. So wissen Sie auch, daß ein unter GEM laufendes Programm diverse Daten für die Erstellung und Verwaltung der einzelnen GEM-Elemente benötigt. Dabei handelt es sich um Koordinatenwerte, Objektnummern, Füllmuster, Darstellungsmodi usw. In der Regel werden diese Daten aus einem separaten Datenfile in das entsprechende GEM-Programm nachgeladen, um dann damit arbeiten zu können.

Um ein Datenfile, *Resource-File* genannt, zu erstellen, benötigen Sie das Resource Construction Set. Bei dem RCS2 handelt es sich um die Version 2.1 des Programms von Digital Research. Viele von Ihnen kennen sicher die ältere Version dieses Programms, das schon seit Jahren auf dem Markt ist. Dieser »Veteran« unter den ST-Programmen wurde jetzt wesentlich verbessert, noch um einige Optionen erweitert und speziell an das GFA-Basic angepaßt.

Nach dem Programmstart präsentiert sich das RCS2 wie im Bild 8.1 dargestellt (hier schon mit einem eingeladenen Resource-File). In einem etwas ungewöhnlich aussehenden Arbeitsfenster sind die *Parts* (Teile, aus denen die Objektbäume und Objekte konstruiert werden) und die Tools (für besondere Einstellungen) sichtbar. Diese können übrigens auch ausgeblendet werden, falls sie stören sollten. Im Pull-down-Menü »Global« befinden sich die entsprechenden Felder.

Die meisten Funktionen des Programms können sowohl mit der Maus als auch mit Tastenkombinationen von [Control] oder [Alternate] aufgerufen werden. Die Arbeit mit dem RCS2 ist somit insgesamt recht einfach.

Natürlich müssen beim Anwender Grundkenntnisse über Aufbau und Hierarchie der Objektstrukturen bestehen, wenn ein Resource-File erstellt werden soll. Für diejenigen unter Ihnen, die bisher noch keinen Gebrauch von einem Resource Construction Set machten, folgt jetzt in Stichworten und kurzen Erläuterungen eine kleine Einführung. Für nähere Informationen gibt es reichlich Literatur. Außerdem finden Sie im GEM-Kapitel dieses Buches weitere Informationen zu Objekten und Objektstrukturen.

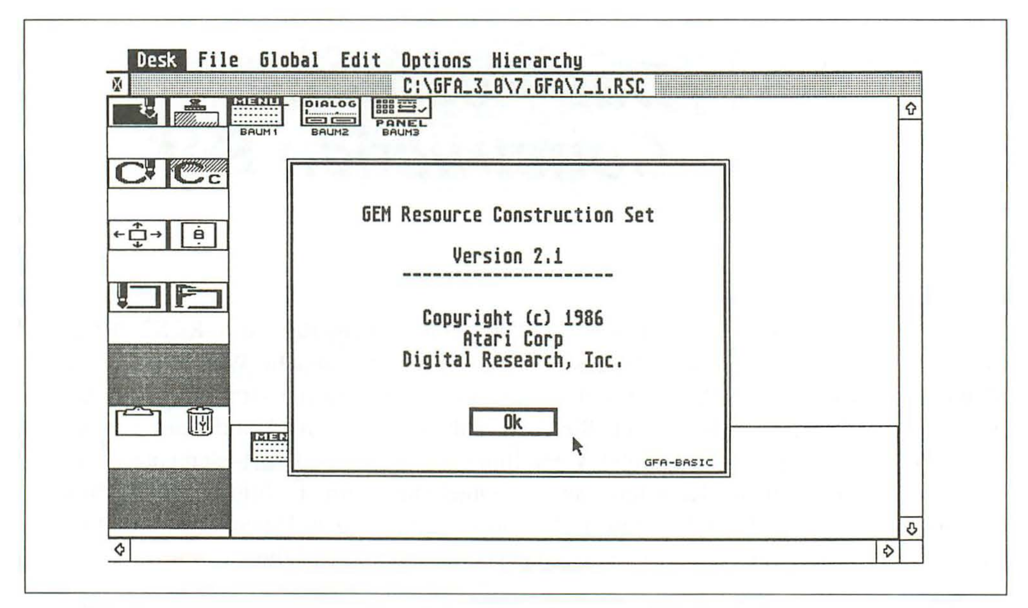


Bild 8.1: Resource Construction Set

Um ein neues Resource zu entwickeln, müssen folgende Punkte beachtet werden.

1. Zunächst muß grundsätzlich mindestens ein *Objektbaum* definiert werden. Dabei handelt es sich um die Grundstruktur einer Menüleiste, einer Dialogbox (hier auch als Panel für besondere grafische Gestaltungen) oder einer Alert-Box.

MENU	Objektbaum für Menüleisten
DIALOG	Objektbaum für Dialogboxen
PANEL	Objektbaum für Dialogboxen
ALERT	Objektbaum für Alarmboxen
FREE	Platzhalter für noch unbekannte Objektbäume

   Schieben Sie einfach das gewünschte Symbol bei gedrückter Maustaste in das Arbeitsfenster. Anschließend muß der Name für diesen Baum angegeben werden. Dabei dürfen Sie auf keinen Fall den Unterstrich »_« eingeben, da es dann schlagartig vorbei ist mit der Herrlichkeit des RCS2; das Programm stürzt nämlich unrettbar ab! Also ist Vorsicht geboten bei der Nameneingabe.

2. Jetzt können Sie durch die Teile des Part-Feldes Ihr Objekt konstruieren. Achten Sie darauf, daß jedes Teil, das später in Ihrem Programm einzeln angesprochen werden muß, einen Namen erhält (kein Unterstrich!). Folgende Teile enthält das Part-Feld:

BUTTON	Ein Feld, das einen Text enthalten und vom Benutzer angeklickt werden kann.
STRING	Eine Zeichenkette.
EDIT	Ein Feld, in das vom Benutzer Text eingegeben bzw. editiert werden kann.
EDIT	Ebenfalls ein editierbares Textfeld, hier jedoch mit einer Linie umrahmt.
HOLLOW BOX	Durchsichtiges Fenster, mit dem andere Objekte eingerahmt werden können.
OPAQUE BOX	Undurchsichtiges Fenster.
TEXT	Text, bei dem Größe, Farbe und Stil eingestellt werden können.
C	Ein einzelnes, nicht editierbares Zeichen innerhalb eines Kastens.
BOXTEXT	Umrahmter Text, bei dem Größe, Farbe und Stil eingestellt werden können.
ICON	Bild, das als Bitmuster gespeichert wird. Das Bild besteht aus einem Vorder- und einem Hintergrundbild.
IMAGE	Wie ICON, jedoch besteht bei diesem Bild kein Hintergrundbild (Maske).

3. Jetzt kann das Resource-File abgespeichert werden. Im Pull-down-Menü File können Sie das entsprechende Feld anklicken und in der Fileselect-Box den gewünschten Namen angeben. Als Extension muß ».RSC« verwendet werden. Es werden nun zwei Dateien gespeichert:

   a) die Resource-Datei (name.RSC)
   b) Definitionsfile für das RCS (name.DFN)

Das Resource-File kann jederzeit wieder eingeladen und bei Bedarf geändert werden. Wichtig ist dazu das Definitionsfile, das immer sicher aufbewahrt werden sollte, denn ohne dieses File, das alle Namen der Objekte enthält, ist das RCS ziemlich aufgeschmissen. Übrigens können Sie mit dem RCS2 auch Resource-Files bearbeiten, die mit dem Resource Construction Set Version 1.x erstellt wurden. Dort hat das Definitionsfile allerdings eine andere Bezeichnung (Extension ».DEF«) und auch einen anderen Aufbau. Ein weiteres Hilfsprogramm auf Ihrer GFA-Basic-Diskette (DEF2DFN.PRG) dient dazu, diese alten Definitionsfiles in neue (Extension ».DFN«) umzuarbeiten.

Soll nun das Resource-File in Ihrem GFA-Basic-Programm genutzt werden, muß es also nachgeladen werden. Auf die dort nötigen Programmschritte möchten wir an dieser Stelle

nicht weiter eingehen. Lesen Sie dazu bitte auch unser GEM-Kapitel. Außerdem arbeiten auch einige unserer Beispielprogramme mit einem Resource-File. Das Resource-File allein reicht in der Regel noch nicht aus. In ihm sind nur die Daten enthalten, um die Objekte zu zeichnen. Was fehlt, sind wieder die Objektnamen. Außerdem muß der Programmierer wissen, welches Objekt welche Objektnummer trägt. Dazu kann eine weitere Datei vom RCS2 gespeichert werden: die Namendatei (Extension ».LST«). Diese Datei ist wirklich sehr wichtig. Sie muß in den Programmtext Ihres Basic-Programms aufgenommen werden. Dazu ist sie als ASCII-Datei gespeichert und kann deshalb einfach mit MERGE übernommen werden. Hier ein Ausschnitt aus der Namendatei zu unserem Programm 7_1.GFA:

```
LET BAUM1&=0            !RSC_TREE
LET TITEL1&=3           !Obj in #0
LET TITEL2&=4           !Obj in #0
LET TITEL3&=5           !Obj in #0
LET INFO&=8             !Obj in #0
LET SPEICHER&=17        !Obj in #0
LET NEU&=18             !Obj in #0
LET ENDE&=20            !Obj in #0
LET LADEN&=22           !Obj in #0
```

Es werden dabei die von Ihnen angegebenen Objektnamen als Basic-Variablen aufgelistet. Diesen Variablen werden die von RCS2 vergebenen Nummern zugewiesen. Jetzt kann jedes Objekt direkt mit einem Variablennamen »angesprochen«, also gezeichnet, bearbeitet, verändert usw. werden. Vorteilhafterweise sollten Sie aus der Variablenliste eine Prozedur machen, die einmal zu Beginn des Programms aufgerufen wird.

Wichtig: Die Namendatei wird vom Resource Construction Set nur dann gespeichert, wenn in der Dialogbox im Pull-down-Menü »Global, Eintrag Output...« das Feld »GFA-Basic« selektiert wurde. Diese Einstellung brauchen Sie aber nur einmal zu machen, da sie mit »Save Preferences« als Voreinstellung gespeichert werden kann und anschließend bei jedem Neustart des RCS2 automatisch voreingestellt wird.

Erläuterungen zu einigen Pull-down-Menü-Einträgen

Menütitel »File«, Eintrag »New«:
Löscht den Arbeitsbereich.

Menütitel »Edit«, Eintrag »Cut«:
Das selektierte Objekt (angeklickt und mit einer gestrichelten Linie umgeben) wird von der Arbeitsfläche auf das Clipboard übernommen.

Menütitel »Edit«, Eintrag »Copy«:
Hier wird das Objekt ebenfalls auf das Clipboard übernommen, jedoch nicht von der Arbeitsfläche entfernt. Der Name des Objekts geht dabei verloren.

Menütitel »Edit«, Eintrag »Paste«:
Holt ein Objekt vom Clipboard.

Menütitel »Edit«, Eintrag »Delete«:
Löscht das Objekt.

Menütitel »Options«, Einträge »Info«, »Name«, »Type«:
Hier erhalten Sie je nach Objekttype verschiedene Informationen, die auch geändert werden können.

Menütitel »Options«, Eintrag »Load«:
Dient zum Einladen der Bitmuster für Icon und Image.

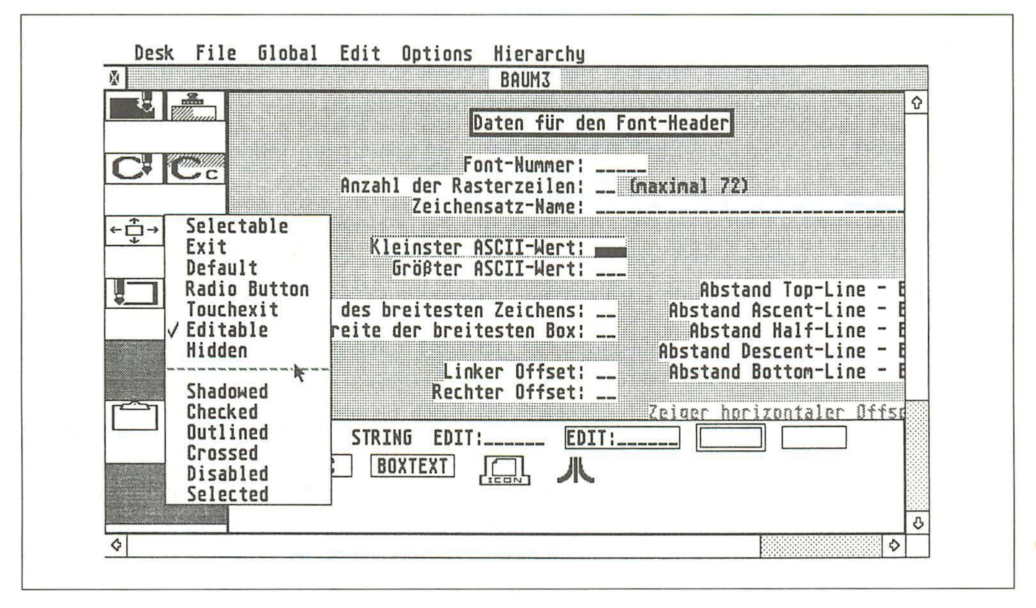


Bild 8.2: Objektflags und Objektstatus

Objektflag und Objektstatus

Folgende Flags und Statusangaben können gemacht werden:

SELECTABLE Das Objekt kann selektiert, d.h., mit der Maus angewählt werden. Das Objekt wird dann invers dargestellt.

EXIT Um eine Objektstruktur, z.B. eine Dialogbox, wieder verlassen zu können, muß mindestens ein Objekt als Exit-Objekt angegeben sein. Siehe auch Touchexit.

DEFAULT Das Objekt kann durch Druck der [Return]-Taste angewählt werden.

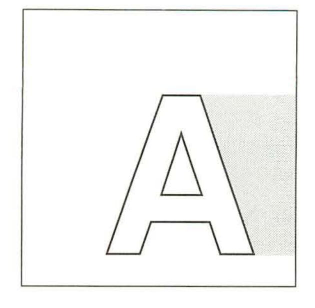
RADIO BUTTON Die Objekte, die in einer Box zusammengefaßt sind, können nur einzeln selektiert werden. Das Selektieren eines Objekts deselektiert

	alle anderen. Siehe z.B. das Resource-File zum Programm 6_3.GFA (Apfelmännchen).
TOUCHEXIT	Das Objekt ist ebenfalls, wie Exit, ein Objekt zum Verlassen des Objektbaums. Hier erfolgt die Reaktion schon beim Anklicken des Objektes, während bei Exit erst beim Loslassen der Maustaste die Objektstruktur verlassen wird. EXIT und TOUCHEXIT dürfen nur einzeln für ein Objekt angegeben werden.
EDITABLE	Das Textfeld dieses Objekts ist änderbar.
HIDDEN	Das Objekt wird ausgeblendet. Es ist jedoch noch im Speicher vorhanden. Diese Objekte können durch Unhide Children im Pull-down-Menü Hierarchy wieder dargestellt werden.
SHADOWED	Das Objekt wird mit einem »Schatten« dargestellt.
CHECKED	Ein sogenannter Checkmarker (Häkchen) wird dargestellt.
OUTLINED	Das Objekt wird mit einer zusätzlichen Linie umrandet.
CROSSED	Das Objekt wird mit zwei diagonalen Linien durchkreuzt.
DISABLED	»Schaltet« ein Objekt ab. Es wird hell dargestellt und kann nicht mehr angewählt werden.
SELECTED	Das Objekt wird vorselektiert. Übrigens, das Exit-Objekt in der Info-Dialogbox des RCS2 wird von unserer Version (2.1) des Programms falsch behandelt. Das Objekt wird nach Verlassen der Dialogbox nicht deselektiert. Dadurch muß der Button beim zweiten Aufruf der Dialogbox doppelt angeklickt werden.

Leider können wir Ihnen an dieser Stelle nicht die ausführliche Bedienungsanleitung zum Resource Construction Set bieten. Laden Sie doch einmal das Resource-File des RCS2 ein (unbedingt vorher eine Sicherheitskopie herstellen!). Es handelt sich hierbei um ein außerordentlich umfangreiches Resource-File. Und hier finden Sie alle Möglichkeiten des Programms wieder. Es läßt sich gut erkennen, aus welchen Objekten die einzelnen Objektbäume bestehen.

Achtung: Speichern Sie das Resource-File nicht wieder ab, kleinste Änderungen könnten es für das Resource Construction Set unbrauchbar machen!

Experimentieren Sie selbst, so lernen Sie schnell die Bedienung des RCS2 kennen. Unsere Programme zeigen Ihnen darüber hinaus, wie die Resource-Files in eigenen Programmen verwendet werden können.



Anhang

Die Befehle des GFA-Basic 3.0

Der Befehlsumfang des GFA-Basic 3.0 ist derart groß, daß eine komplette ausführliche Befehlsbeschreibung in diesem Buch leider nicht erfolgen kann. Wir haben uns dennoch zu einer Art Kurzbeschreibung durchgerungen, da wir dieses Buch auch als ein Nachschlagewerk ansehen. Oftmals ist es so, daß man bei der Programmierung »nur mal eben« die genaue Syntax eines Befehls benötigt, obwohl einem die Wirkung dieses Befehls vollkommen klar ist. Hier kann unsere Auflisting der Befehle sicher gute Dienste leisten. Dazu haben wir nämlich die genaue Schreibweise aller Befehle mit einer jeweils sehr kurzen Funktionsbeschreibung in diesen Anhang A aufgenommen.

Damit das Suchen nach bestimmten Befehlen oder Funktionen einfacher ist, ist die Liste in Funktionsgruppen aufgeteilt. Wir haben uns auch bemüht, im Stichwortverzeichnis am Ende des Buches alle Befehle aufzunehmen. Damit müßte es recht einfach sein, bestimmte Befehle herauszufinden.

In der Beschreibung der Befehlssyntax halten wir uns hauptsächlich an die Schreibweise in der GFA-Basic-Anleitung. Oftmals setzten wir jedoch auch eigene Bezeichnungen der Parameter ein, wenn dies der Erläuterung der Syntax dienlich war. Die Befehle innerhalb der Funktionsgruppen sind in alphabetischer Reihenfolge aufgeführt.

Der Anhang A ist in folgende Bereiche aufgeteilt:

Variablen- und Speicherverwaltung	ab Seite 258
Reservierte Variablen	ab Seite 266
Logische Operatoren	ab Seite 267
Numerische Funktionen	ab Seite 267
Zeichenkettenverwaltung	ab Seite 274
Ein- und Ausgabefunktionen	ab Seite 276
Programmsteuerung	ab Seite 287
Grafik	ab Seite 295
Line-A-Routinen	ab Seite 299
Ereignis-, Menü- und Fensterverwaltung	ab Seite 300
Systemroutinen	ab Seite 303
AES-Bibliotheken	ab Seite 307

Variablen- und Speicherverwaltung

$ Text

Die Angabe »$« am Zeilenanfang einer Programmzeile dient der Kennzeichnung eines Compilerbefehls.

wert={adresse}
{adresse}=wert

Liest bzw. schreibt einen 4-Byte-Integerwert »wert« an die angegebene »adresse«.

ABSOLUTE var1,*var2

Der Variablenpointer der Variablen »var1« erhält die Anfangsadresse der Variablen »var2« (siehe auch ARRPTR, VARPTR).

ARRPTR(var)
*var

Ermittelt die Adresse einer Variablen. Bei Variablenfeldern oder Strings erhält man die Adresse des Deskriptors (siehe auch VARPTR).

ARRAYFILL var(),wert

Alle Elemente eines Arrays erhalten den »wert«.

ASC(string$)

Der ASCII-Code des ersten Zeichens im »string$« wird ermittelt.

BASEPAGE

Hierdurch kann die Basepage des GFA-Basic ermittelt werden.

BIN$(zahl[,anzahl])

Die »zahl« wird in eine Binärzahl mit »anzahl« (1 bis 32) Stellen umgewandelt. Das Ergebnis ist ein String.

BMOVE quelle,ziel,anzahl

Mit BMOVE kann ein Speicherbereich (»anzahl« Bytes) mit der Anfangsadresse »quelle« in den Bereich mit der Anfangsadresse »ziel« kopiert werden. Die Bereiche dürfen sich auch überlappen.

wert=BYTE{adresse}
BYTE{adresse}=wert

Liest bzw. schreibt ein Byte »wert« an die »adresse«.

wert=CARD{adresse}
CARD{adresse}=wert

Liest bzw. schreibt einen 2-Byte-Integerwert »wert« an die »adresse«.

string$=CHAR{adresse}
CHAR{adresse}=string$

Liest bzw. schreibt einen mit einem Null-Byte endenden String »string$« an die »adresse«.

CHR$(code)

Bildet ein Zeichen mit dem angegebenen ASCII-Code.

CFLOAT(wert)

Die angegebene Zahl »wert« wird in eine Fließkommazahl umgewandelt.

CINT(wert)

Die angegebene Zahl »wert« wird in eine gerundete Integerzahl umgewandelt.

CLEAR

Löscht alle Variablen und Felder. CLEAR wird beim Start eines Programms automatisch durchgeführt.

CLR var1[,var2...]

Die angegebenen Variablen (keine Strings) werden gelöscht. Siehe auch ERASE.

CVD(string$)

Die 8-Byte-Zeichenkette in »string$« wird in eine Fließkommazahl umgewandelt (siehe auch MKD$).

CVF(string$)

Wandelt eine 6-Byte-Zeichenkette in eine Fließkommazahl um (siehe auch MKF$).

CVI(string$)

Wandelt eine 2-Byte-Zahlenkette in eine 2-Byte-Integerzahl um (siehe auch MKI$).

CVL(string$)

Wandelt eine 4-Byte-Zeichenkette in eine 4-Byte-Integerzahl um (siehe auch MKL$).

CVS(string$)

Eine 4-Byte-Zeichenkette, die einen Wert im ST-Basic-Format (nicht OMIKRON!) enthält, wird in eine Fließkommazahl umgewandelt (siehe auch MKS$).

DATE$
DATE$=string$

DATE$ ermittelt das Systemdatum im je nach Modus eingestellten Format. Es kann auch das Datum neu eingestellt werden. Dabei muß ebenfalls das Format beachtet werden, da die Änderung sonst nicht übernommen wird.

DEFBIT String$
DEFBYT String$
DEFWRD String$
DEFINT String$
DEFFLT String$ (entspricht: **DEFDBL und DEFSNG**)
DEFSTR String$

Variablendeklaration. Hierbei werden alle Variablen, deren Anfangsbuchstaben im »String$« angegeben sind, bestimmten Variablentypen zugeordnet. Somit kann man im weiteren Programmverlauf auf die Postfixes verzichten.

DEFLIST Wert

Hierdurch wird die Schreibweise der Befehle, Funktionen und Variablen der GFA-Basic-Listings festgelegt.

DELETE feld(element)

Das »element« im Variablenfeld »feld« wird gelöscht. Alle darauffolgenden Elemente rücken um eine Stelle auf (siehe auch INSERT).

DIM var(dimension)

Variablenfelder müssen zu Programmbeginn mit diesem Befehl dimensioniert werden. Es können auch mehrere Felder beliebiger Variablentypen mit einem Befehl dimensioniert werden.

DIM?(var())

Ermittelt die Anzahl der Elemente eines Variablenfeldes.

wert=DOUBLE{adresse}

DOUBLE{adresse}=wert

Liest bzw. schreibt einen 8-Byte-Fließkommawert im IEEE-Double-Format an die »adresse«.

DPEEK(adresse)

Liest zwei Byte ab »adresse«.

DPOKE adresse,wert

Schreibt »wert« an die »adresse«.

ERASE feld()

Mit ERASE können eines oder mehrere Variablenfelder gelöscht werden. Anschließend können die Felder auch neu dimensioniert werden.

FALSE

Systemvariable mit dem Wert 0 für »logisch falsch«.

wert=FLOAT{adresse}
FLOAT{adresse}=wert

Liest bzw. schreibt einen 8-Byte-Fließkommawert (GFA-3.0-Format) an die »adresse«.

FRE()
FRE(x)

Der noch freie Speicherplatz wird ermittelt. Dabei kann auch eine Garbage-Collection erzwungen werden, wenn ein beliebiger Ausdruck »x« in der Klammer angegeben wird.

HEX$(zahl[,anzahl])

Die »zahl« wird in eine Hexadezimalzahl (Sedezimalzahl) mit »anzahl« (1 bis 8) Stellen umgewandelt. Das Ergebnis ist ein String.

HIMEM

Ermittelt die Endadresse +1 des vom GFA-Basic genutzten Speichers.

INLINE adresse,anzahl

Dieser Befehl reserviert einen Speicherbereich mit »anzahl« (<32700) Byte und liefert die Anfangsadresse »adresse« zurück. Der Speicherbereich wird mit Null-Bytes gefüllt. Beim Abspeichern, bzw. Einladen des Programms wird dieser Speicherbereich jeweils mitgespeichert bzw. mitgeladen.

INSERT var(element)=wert

Dieser Befehl fügt an der angegebenen Stelle »element« des Variablenfeldes einen »wert« ein. Alle folgenden Elemente des Feldes rücken um eine Position nach hinten. Das letzte Element geht dabei verloren.

wert=INT{adresse}
INT{adresse}=wert

Liest bzw. schreibt einen 2-Byte-Integerwert (mit Vorzeichen) an die »adresse«. Entspricht WORD{ }.

LET

LET erlaubt eine Variablenzuweisung. Der Befehl kann auch weggelassen werden. Er ermöglicht allerdings auch die Zuweisung an Variablen, die z.B. Namen von Basic-Befehlen tragen.

wert=LONG{adresse}
LONG{adresse}=wert

Liest bzw. schreibt eine 4-Byte-Integerzahl an die angegebene »adresse«.

LPEEK(adresse)

Liest 4 Byte ab der angegebenen »adresse«.

LPOKE adresse,wert

Schreibt einen 4-Byte-Wert »wert« an die »adresse«.

adresse=MALLOC(anzahl)

Es werden »anzahl« Bytes Speicher reserviert. Dabei wird die Anfangsadresse zurückgeliefert. Hat »anzahl« den Wert –1, ermittelt diese Funktion die Anfangsadresse des größten zusammenhängenden freien Speicherbereiches.

MFREE(adresse)

Gibt den zuvor mit MALLOC reservierten Speicherbereich wieder frei.

MSHRINK(adresse,anzahl)

Der zuvor mit MALLOC reservierte Speicherbereich wird auf die neuen Daten verkleinert.

OCT$(zahl[,anzahl])

Wandelt die angegebene »zahl« in eine Oktalzahl (Zahlensystem zur Basis 8) mit »anzahl« (1 bis 11) Stellen.

OPTION BASE 0
OPTION BASE 1

Je nach verwendetem Befehl enthalten Felder als kleinstes Element das Element 0 oder 1. Standardeinstellung bei Programmbeginn ist 0.

PEEK(adresse)

Liest ein Byte an der »adresse«.

PI

Systemvariable, die die Kreiszahl PI (3,14...) enthält.

POKE adresse,wert

Schreibt ein Byte (»wert«) an die »adresse«.

QSORT var([+-])[,anzahl][,var2%()]
SSORT var([+-])[,anzahl][,var2%()]
QSORT var$([+-])[OFFSETz][WITH intvar()][,anzahl[,var2%()]]
SSORT var$([+-])[OFFSETz][WITH intvar()][,anzahl[,var2%()]]

Diese Befehle dienen zum Sortieren beliebiger Variablenfelder. Dabei sortiert QSORT nach dem Quicksort-Verfahren und »SSORT« nach dem Shellsort-Verfahren. Mit der Befehlserweiterung »WITH« kann auch ein Sortierkriterium angegeben werden. Ansonsten wird nach der ASCII-Tabelle verfahren. Wahlweise kann ein zweites Variablenfeld mitsortiert werden. Ab GFA-Basic Version 3.02 kann mit »OFFSET« angegeben werden, wie viele Zeichen (»z«) ab Stringanfang bei der Sortierung nicht beachtet werden sollen.

RESERVE[anzahl]

Stellt den vom GFA-Basic genutzten Speicherbereich auf die angegebene »anzahl« (Angabe in 256er-Schritten) Byte. Wird die Angabe weggelassen, wird der beim Start des Interpreters eingestellte Zustand wiederhergestellt.

SDPOKE adresse,wert

Schreibt an die »adresse« 2 Byte. Dies geschieht im Supervisormodus, bei dem auch sonst geschützte Speicherbereiche überschrieben werden können.

SETTIME zeit$,datum$

Ersetzt die Befehle »TIME$=« und »DATE$=«.

wert=SINGLE{adresse}
SINGLE{adresse}=wert

Liest bzw. schreibt eine 4-Byte-Fließkommazahl im IEEE-Single-Format an die »adresse«.

SLPOKE adresse,wert

Schreibt im Supervisormodus einen 4-Byte-Wert an die »adresse«.

SPOKE adresse,wert

Schreibt im Supervisormodus ein Byte an die »adresse«.

SSORT...

Siehe QSORT.

STR$(zahl)
STR$(zahl,stellen)
STR$(zahl,stellen,anzahl)

Dieser Befehl wandelt eine »zahl« in einen String. Dabei kann mit »stellen« angegeben werden, wie viele Zeichen dieser String erhalten soll. Wird zusätzlich »anzahl« angegeben, so wird der String mit »anzahl« Nachkommastellen versehen.

SWAP var1,var2
SWAP feld1(),feld2()
SWAP *feld1_adresse,feld2()

SWAP vertauscht die Inhalte zweier Variablen oder die Inhalte zweier Felder (hier erfolgt eine automatische Dimensionierung). Die dritte Syntax dient dazu, einer Prozedur ein Feld zu übergeben, indem nur die Deskriptoradresse dieses Feldes angegeben wird.

TIME$
TIME$=zeit$

Ermittelt die Systemzeit (HH:MM:SS) bzw. stellt die Systemuhr neu ein.

TIMER

Diese Systemvariable enthält die seit dem Einschalten des Systems vergangene Zeit in 200stel-Sekunden.

TRUE

Systemvariable, die den Wert für logisch wahr (–1) enthält.

TYPE(*variable)

Ermittelt den Typ der angegebenen Variablen. Das Ergebnis ist ein Wert von –1 bis 13.

V:x

Abkürzung für VARPTR(x).

VAL(string$)
VAL?(string$)

Hierdurch wird ein String in eine Zahl umgewandelt. Dies geschieht nur so lange, wie die einzelnen Zeichen (beginnend beim ersten Zeichen von links) auch Ziffern entsprechen. Auch Vorzeichen und Dezimalpunkt werden übersetzt.

VARPTR(x)

Ermittelt die Adresse der Variablen. Bei Strings oder Feldern wird die Deskriptoradresse zurückgegeben.

VOID

VOID kann als Rückgabewert beim Aufruf einer Funktion eingesetzt werden. Die Funktion liefert dann einen Wert zurück, der dann »vergessen« wird.

WORD{}

Siehe INT{ }.

~

Die Tilde hat die gleiche Wirkung wie VOID. Allerdings erfolgt hier keine Umrechnung in Fließkomma (dadurch schneller).

Reservierte Variablen

DATE$

Enthält das Systemdatum im Format TT.MM.JJJJ oder MM/TT/JJJJ, je nach eingestelltem Modus (mit MODE).

FALSE

Enthält den Wert für »logisch Falsch« (0).

PI

Enthält den Wert der Kreiszahl Pi.

TIMER

Enthält die seit dem Einschalten des Systems vergangene Zeit in 200stel-Sekunden.

TIME$

Enthält die Systemzeit im Format HH.MM.SS.

TRUE

Enthält den Wert für »logisch Wahr« (–1).

Logische Operatoren

NOT x

NOT negiert den logischen Ausdruck x.

x AND y

Logische UND-Verknüpfung von x und y.

x OR y

Logische ODER-Verknüpfung von x und y.

x XOR y

Logische Exklusiv-ODER-Verknüpfung von x und y.

x IMP y

Implikation von x und y.

x EQV y

Bildet die logische Äquivalenz von x und y.

Numerische Funktionen

ABS(x)

Liefert den ganzzahligen Teil von x.

ACOS(winkel)

Berechnet den Arcuscosinus des im Bogenmaß angegebenen Winkels.

ADD x,y

Entspricht: x = x + y

z=ADD(x,y)

Entspricht: z = x + y (Integerarithmetik).

z=AND(x,y)

Entspricht: z = x AND y

ASIN(winkel)

Berechnet den Arcussinus des im Bogenmaß angegebenen Winkels.

ATN(winkel)

Berechnet den Arcustangens des im Bogenmaß angegebenen Winkels.

BCHG(x,y)

Das Bit y im Wert x wird negiert.

BCLR(x,y)

Löscht (0) das Bit y im Wert x.

BSET(x,y)

Setzt (1) das Bit y im Wert x.

BTST(x,y)

Ermittelt, ob das Bit y im Wert x gesetzt ist. Ergebnis ist dann TRUE (–1). Ist das Bit nicht gesetzt, wird FALSE (0) zurückgeliefert.

BYTE(x)

Liefert die unteren 8 Bit des Wertes x zurück.

CARD(x)

Liefert die unteren 16 Bit des Wertes x zurück.

COS(winkel)

Berechnet den Cosinus des im Bogenmaß angegebenen Winkels.

COSQ(grad)

Ermittelt den Cosinus des Winkels »grad« anhand einer Tabelle. Dabei werden die Zwischenwerte zwischen zwei Graden in Sechzehntel-Schritten linear interpoliert, was eine geringere Genauigkeit als COS ergibt, aber wesentlich schneller ist.

DEC x

Entspricht: x = x – 1

grad=DEG(bogenmass)

Errechnet das Gradmaß des angegebenen Bogenmaßes (Umkehrfunktion zu RAD).

DIV x,y

Entspricht: x = x / y

z=DIV(x,y)

Entspricht: z = x / y (Integerarithmetik).

z=x DIV y

Ermittelt das Ergebnis der Division von x durch y.

z=EQV(x,y)

Bildet in z die Äquivalenz von x und y.

EVEN(x)

Ermittelt, ob der Wert x gerade ist (Ergebnis = TRUE), oder ungerade (Ergebnis = FALSE).

EXP(x)

Ermittelt die x-te Potenz zur Basis der Zahl e.

FIX(x)

Gibt den ganzzahligen Teil von x zurück. Schneidet den Nachkommateil der Zahl x ab. Diese Funktion entspricht TRUNC.

FRAC(x)

Gibt den Nachkommateil von x zurück.

z=IMP(x,y)

Bildet in z die Implikation von x und y.

INC x

Entspricht: x = x + 1

INT(x)

Gibt bei positivem x den ganzzahligen Teil von x zurück. Bei ungeraden negativen Zahlen wird die nächste kleinere ganze Zahl zurückgegeben.

LOG(x)

Ermittelt den Logarithmus der Zahl x zur Basis e.

LOG10(x)

Ermittelt den Logarithmus der Zahl x zur Basis 10.

MAX(werte)
MAX(strings$)

Ermittelt den größten Wert der in der Klammer angegebenen Werte-Liste. Es können auch mehrere Strings angegeben werden. Dann ermittelt die Funktion nach ASCII-Codes den »größten« String.

MIN(werte)
MIN(strings$)

Ermittelt den kleinsten Wert. Sonst wie MAX.

z=MOD(x,y)

Entspricht: z = x MOD y (z enthält den Restwert bei der Division von x und y).

x MOD y

Ermittelt den Rest bei der Division von x durch y.

MUL x,y

Entspricht: x = x * y

z=MUL(x,y)

Entspricht: z = x * y (Integerarithmetik).

ODD(x)

Ermittelt, ob x ungerade (Ergebnis = TRUE) oder gerade ist (Ergebnis = FALSE).

z=OR(x,y)

z erhält das Ergebnis einer logischen ODER-Verknüpfung von x und y.

PRED(zahl)

Liefert den nächstkleineren Wert der »zahl« zurück (siehe auch SUCC und PRED(string$)).

RAD(grad)

Ermittelt aus dem im Gradmaß angegebenen Grad das Bogenmaß (siehe auch DEG).

RAND(zahl)

Hierdurch kann eine Zufallszahl (16-Bit-Zahl zwischen 0 und Zahl – 1) gebildet werden.

RANDOM(zahl)

Bildet eine ganzzahlige Zufallszahl zwischen 0 und Zahl – 1.

RANDOMIZE wert

Hierdurch erhält der Zufallszahlengenerator einen Startwert »wert«. Bei jeweils gleichem Wert ergeben die folgenden Zufallszahlen auch immer die gleiche Folge von Werten. Man kann so z.B. Daten mittels einer »Codezahl« ver- und entschlüsseln.

RND[(zahl)]

Ergibt Zufallszahlen zwischen 0 und <1. Auf die Angabe des Parameters »zahl« kann verzichtet werden.

z=ROL(x,y)
z=ROL&(x,y)
z=ROL|(x,y)

Verschiebt den Inhalt von x um y Bit nach links. Dabei werden die links hinausgeschobenen Bits jeweils wieder rechts in x aufgenommen. Ohne Angabe wird x als Langwort (4 Byte) angenommen, mit »&« als Wort (2 Byte) und mit »|« als Byte. Das Ergebnis der Funktion erscheint in z.

z=ROR(x,y)
z=ROR&(x,y)
z=ROR|(x,y)

Verschiebt den Inhalt von x um y Bit nach rechts, sonst wie ROL.

ROUND(x[,stellen])

Ermittelt einen auf »stellen« gerundeten Wert von x. Ohne Angabe von »stellen« oder »stellen« = 0 wird auf ganze Zahlen gerundet. Ist »stellen« ein negativer Wert, so wird vor dem Komma gerundet.

SGN(zahl)

Ermittelt das Vorzeichen der Zahl.

z=SHL(x,y)
z=SHL&(x,y)
z=SHL|(x,y)

Hierdurch wird der Inhalt von x um y Stellen nach links verschoben. Dabei werden jeweils Bits von rechts nachgeschoben. Das Ergebnis wird in z übergeben.

z=SHR(x,y)
z=SHR&(x,y)
z=SHR|(x,y)

Hierdurch wird der Inhalt von x um y Stellen nach rechts verschoben. Sonst wie SHL.

SIN(winkel)

Berechnet den Sinus des im Bogenmaß angegebenen Winkels.

SINQ(grad)

Berechnet den Sinus des im Gradmaß angegebenen Winkels »grad«. Wesentlich schneller als SIN. Siehe auch COSQ.

SQR(zahl)

Berechnet die Quadratwurzel der »zahl«.

SUB x,y

Entspricht: x = x – y

z=SUB(x,y)

Übergibt z das Ergebnis der Subtraktion von x und y (Integerarithmetik).

SUCC(zahl)

Ermittelt den nächstgrößeren Wert von »zahl«. Siehe auch PRED und SUCC(string$).

SWAP(zahl)

Diese Funktion vertauscht das untere und obere Wort von »zahl«.

TAN(winkel)

Berechnet den Tangens des im Bogenmaß angegebenen Winkels.

TRUNC(x)

Gibt den ganzzahligen Teil von x zurück. Schneidet den Nachkommateil der Zahl x ab. Diese Funktion entspricht FIX.

WORD(x)

Erweitert den Wert x auf Langwortlänge (4 Byte).

z=XOR(x,y)

Übergibt z das Ergebnis der logischen Exklusiv-ODER-Verknüpfung von x und y.

Zeichenkettenverwaltung

INSTR(a$,b$,[x])

Untersucht, ob im String »a$« die Zeichenfolge »b$« enthalten ist und gibt die Anfangsposition zurück. Wird »b$« nicht gefunden, ist das Ergebnis 0. Die Suche kann wahlweise auch beim x-ten Zeichen in »a$« beginnen. Siehe auch RINSTR.

LEFT$(a$[,x])

Gibt die ersten x Zeichen des Strings »a$« zurück. Wird x weggelassen, übergibt die Funktion nur das erste Zeichen von »a$«.

LEN(a$)

Diese Funktion ermittelt die Anzahl der Zeichen in »a$«.

LSET a$=b$

Hierdurch wird der Inhalt von »b$« linksbündig in »a$« übernommen. Dabei werden zusätzliche Leerzeichen eingesetzt, wenn »a$« länger ist als »b$«.

MID$(a$,x[,y])

Diese Funktion übergibt die y Zeichen ab dem x-ten Zeichen aus »a$«. Wird y weggelassen, werden alle Zeichen ab dem x-ten Zeichen übergeben.

MID$(a$,x[,y])=b$

Übergibt dem String »a$« ab dem x-ten Zeichen y Zeichen des Strings »b$«.

MKD$(zahl)

Wandelt eine Fließkommazahl des GFA-Basic 3.0 in einen 8-Byte-String. Umkehrfunktion von CVD.

MKF$(zahl)

Wandelt eine Fließkommazahl des GFA-Basic 1.0 und 2.0 in einen 6-Byte-String. Umkehrfunktion von CVF.

MKI$(zahl)

Wandelt eine 16-Bit-Integerzahl in einen 2-Byte-String. Umkehrfunktion von CVI.

MKL$(zahl)

Wandelt eine 32-Bit-Integerzahl in einen 4-Byte-String. Umkehrfunktion von CVI.

MKS$(zahl)

Wandelt eine Fließkommazahl in eine dem ST-Basic (nicht OMIKRON) kompatible Form und schreibt dieses Format in einen 4-Byte-String. Umkehrfunktion von CVS.

PRED(a$)

Ermittelt den ASCII-Code des ersten Zeichens in a$ und gibt diesen Wert –1 zurück.

RIGHT$(a[,x])

Gibt die letzten x Zeichen des Strings »a$« zurück. Wird x weggelassen, übergibt die Funktion nur das letzte Zeichen von »a$«.

RINSTR(a$,b$,[x])

Untersucht, ob im String »a$« die Zeichenfolge »b$« enthalten ist, und gibt die Anfangsposition zurück. Wird »b$« nicht gefunden, ist das Ergebnis 0. Hierbei beginnt die Suche am Ende des »a$«. Siehe auch INSTR.

RSET a$=b$

Hierdurch wird der Inhalt von »b$« rechtsbündig in »a$« übernommen. Dabei werden zusätzliche Leerzeichen eingesetzt, wenn »a$« länger ist als »b$«.

SPACE$(anzahl)

Diese Funktion bildet einen Leerstring bestehend aus Anzahl Space.

SPC(anzahl)

Diese Funktion gibt bei einem PRINT-Befehl »anzahl« Space aus.

STRING$(anzahl,a$)
STRING$(anzahl,ascii_code)

Hierdurch kann ein String bestehend aus »anzahl * a$« oder aus »anzahl« Zeichen mit dem ASCII-Code erzeugt werden.

SUCC(a$)

Ermtittelt den ASCII-Code des ersten Zeichens in »a$« und gibt diesen Wert +1 zurück.

TRIM$(a$)

Entfernt die Leerzeichen am Anfang und Ende des Strings »a$«.

UPPER$(a$)

Hierdurch werden alle Kleinbuchstaben eines Strings in Großbuchstaben umgewandelt.

Ein- und Ausgabefunktionen

BGET #n,adresse,anzahl

BGET lädt aus einer zuvor geöffneten Datei mit der Kanalnummer »n« die »anzahl« Bytes (ab der momentanen Position des Filepointers) und speichert diese an der Anfangsadresse »adresse« im Speicher ab.

BLOAD name$[,adresse]

Lädt die Datei mit »namen$« (komplette Pfadangabe) an die »adresse«. Wird diese Angabe weggelassen, wird die zuletzt bei BSAVE genutzte Adresse angenommen.

BPUT #n,adresse,anzahl

Speichert den Speicherbereich ab »adresse« mit »anzahl« Bytes in die zuvor geöffnete Datei mit der Kanalnummer »n«.

BSAVE name$,adresse,anzahl

Speichert den Speicherbereich ab »adresse« mit »anzahl« Bytes in die Datei »namen$«.

CLOSE[#n]

Hierdurch wird die Datei mit der Kanalnummer »n« geschlossen. Wenn die Angabe weggelassen wird, werden alle zuvor eventuell geöffneten Dateien geschlossen.

CHDIR name$

CHDIR legt einen aktuellen Zugriffspfad neu mit dem »namen$« fest. Dieser kann dann zum Beispiel von DIR$ wieder angezeigt werden.

CHDRIVE n
CHDRIVE n$

Legt das aktuelle Laufwerk fest. In »n« ist die Laufwerknummer (1 bis 16) enthalten. Alternativ dazu kann im String a$ der Kennbuchstabe des Laufwerks angegeben werden.

CRSCOL

Ermittelt die aktuelle Spaltenposition (Links ist 1) des Cursors.

CRSLIN

Ermittelt die aktuelle Zeilenposition (Oben ist 1) des Cursors.

DATA daten1[;daten2...]

Hinter einem DATA können diverse Werte angegeben werden (Strings oder numerische Angaben), die durch Kommas getrennt werden (sogenannte DATA-Zeile). Diese können dann durch READ eingelesen werden.

DEFNUM anzahl

Legt fest, mit wie vielen Stellen (anzahl) Zahlen durch den PRINT-Befehl ausgegeben werden sollen.

DFREE(n)

Ermittelt den noch freien Bereich eines Laufwerks mit der Laufwerkkennung »n«.

DIR pfad$[TO name$]

Hierdurch wird das Inhaltsverzeichnis des angegebenen »pfad$« auf dem Bildschirm oder über den Dateinamen »name$« ausgegeben.

DIR$(n)

Diese Funktion ermittelt den aktuellen Zugriffspfad des Laufwerks »n« oder des aktuellen Laufwerks, wenn n = 0.

EOF(#n)

Diese Funktion liefert als Ergebnis TRUE, wenn der Filepointer am Dateiende der Datei mit der Filenummer »n« steht.

EXIST(name$)

Durch EXIST kann festgestellt werden, ob eine bestimmte Datei mit »namen$« vorhanden ist (TRUE, falls vorhanden).

FGETDTA()

Diese Funktion liefert die »disk transfer address« zurück.

FIELD #n,anzahl AS string$[,anzahl2 AS string2$...]

Dient zum Einrichten einer relativen Datei. Hierbei muß für jedes Feld in den Datensatz die Länge angegeben werden. Bei der anschließenden Arbeit mit der Datei müssen diese voreingestellten Längen stets eingehalten werden.

FIELD #n,anzahl AT (var)[,anzahl AT (var2)...]

Diese besondere Variante des FIELD-Befehls dient ebenfalls dazu, einen Datensatz in Felder zu unterteilen. Allerdings wird hierbei jeweils ein Zeiger auf einen Speicherbereich (das sollten Variablenpointer sein) und die Anzahl Bytes, die gespeichert werden sollen, angegeben. Die zwei Befehlsvarianten können auch gemischt angegeben werden.

FILES pfadname$ [TO name$]

Siehe DIR-Befehl. Zusätzlich werden bei FILES das Datum, die Uhrzeit und die Länge jeder Datei ausgegeben.

FORM INPUT anzahl,a$
FORM INPUT anzahl AS a$

Die formatierte Eingabe von Zeichen von der Tastatur in einen String. Dabei werden »anzahl« (von 1 bis 255) Zeichen übernommen. Ein Carriage Return beendet die Eingabe. Bei der zweiten Befehlsvariante wird zuvor der Inhalt von a$ auf dem Bildschirm ausgegeben.

FSETDTA(adresse)

Stellt eine neue »disk transfer address« ein. Mit Vorsicht behandeln!

FSFIRST(pfad$,attribut)

Ermittelt die erste Datei, die mit »pfad$« erreicht wird und die die angegebenen Attribut-Bits in der DTA enthält.

FSNEXT()

Sucht die nächste Datei, die die Kriterien von FSFIRST erfüllt.

GET #n[,satz]

Mit GET wird der nächste Datensatz bzw. der Datensatz mit der Nummer »satz« aus einer relativen Datei gelesen.

HARDCOPY

Durch diesen Befehl wird die Hardcopy-Funktion (wie durch Druck der Tasten `Alternate` + `Help`) gestartet, die den Bildschirminhalt auf dem Drucker ausgibt.

HIDEM

Die Bildschirmdarstellung des Mauscursors wird abgeschaltet, jedoch nicht die Mausabfrage.

HTAB spalte

Positioniert den Cursor an die angegebene Spalte.

INKEY$

INKEY$ liest ein Zeichen von der Tastatur, wartet jedoch nicht auf das Drücken einer Taste.

INP(n)

Diese Funktion liest ein Byte von der angeschlossenen Peripherie. Dabei wird so lange gewartet, bis ein Zeichen verfügbar ist. Mit n kann ein Wert von 0 bis 5 für bestimmte Geräte angegeben werden oder eine beliebige Dateinummer.

INP?(n)

Hierdurch wird der Eingabestatus eines Peripheriegerätes überprüft (FALSE, wenn das Gerät nicht bereit ist).

INP(#n)

Liest ein Byte aus der Datei mit der Dateinummer »n«.

INP&(#n)
INP%(#n)

Liest 16 Bit bzw. 32 Bit aus der Datei (erst ab Version 3.04). Siehe auch OUT& und OUT%.

INPAUX$

Diese Funktion dient zum schnellen Einlesen von Daten über die serielle Schnittstelle.

INPMID$

Diese Funktion dient zum schnellen Einlesen von Daten über die MIDI-Schnittstelle.

INPUT["text";]werte

Dieser Eingabebefehl erwartet eine Eingabe über die Tastatur. Dabei kann auch ein Text mitausgegeben werden. Die Eingabe wird durch [Return] beendet. »werte« kann aus mehreren Variablen oder Strings bestehen, die durch Kommas getrennt sind und auch durch Kommas getrennt eingegeben werden können.

INPUT#n,var1[,var2...]

Liest Daten aus einer sequentiellen Datei.

INPUT$(anzahl,[#n])

Liest »anzahl« Zeichen von der Tastatur oder aus der Datei mit der Dateinummer »n«.

KEYDEF f_taste,string$

Belegt die Funktionstaste »f_taste« (1 bis 20) mit dem »string$« (maximal 31 Zeichen).

KEYGET n

Diese Funktion wartet auf einen Tastendruck. In »n« wird dann ein Langwort, bestehend aus dem ASCII-Code, dem Scan-Code und dem Status der Tastatur-Umschalttasten zurückgegeben.

KEYLOOK n

KEYLOOK liest Daten aus dem Tastaturpuffer, ohne dessen Inhalt zu verändern.

KEYPAD n

Mit KEYPAD wird der Status der Tastatur-Umschalttasten eingestellt.

KEYPRESS n

Durch KEYPRESS wird das Drücken einer Taste simuliert. In »n« steht dabei der ASCII-Code der [Taste].

KEYTEST n

Liest ein Zeichen von der Tastatur, wartet jedoch nicht auf das Drücken einer Taste.

KILL name$

Löscht die Datei »name$«.

LINE INPUT["text";]werte

Dieser Befehl arbeitet wie INPUT. Hier können allerdings auch Kommas eingegeben werden.

LINE INPUT #n,string$[,string2$...]

Hierdurch werden aus einer Datei sequentiell Zeichenketten eingelesen.

LOCATE zeile,spalte

Positioniert den Cursor an die angegebene Stelle.

LOC(#n)

Ermittelt die aktuelle Position des Filepointers der Datei mit der Dateinummer »n«.

LOF(#n)

Ermittelt die Länge der Datei mit der Dateinummer »n«.

LPOS(x)

Ermittelt die seit dem letzten Carriage Return an den Drucker ausgegebenen Zeichen.

LPRINT werte

Hierdurch werden Daten über den Drucker ausgegeben. Der Befehl wirkt wie PRINT.

MKDIR name$

Hierdurch wird ein Ordner (Inhaltsverzeichnis) mit dem Pfadnamen »name$« angelegt.

MODE n

Bestimmt den Ausgabemodus (n = 0 bis 3) für USING und DATE$.

MOUSE x,y,k

Liefert die Koordinaten der Maus sowie den Status der Maustasten. Diese Daten können mit den folgenden Befehlen auch einzeln abgefragt werden.

MOUSEK

Liefert den Status der Maustasten zurück.

MOUSEX
MOUSEY

Ermittelt die X- bzw. Y-Position der Maus.

NAME alt$ AS neu$

Hierdurch erhält die Datei mit Namen »alt$« den Namen »neu$«.

OPEN "modus",#n,name$[,satzlaenge]

Mit OPEN wird ein Datenkanal geöffnet und erhält die Dateinummer »n«. Bei relativen Dateien muß die Länge der Datensätze mit »satzlaenge« angegeben werden.

OUT[#]n,wert[,wert2...]

Gibt ein oder mehrere Bytes (wert) an den Datenkanal »n« aus.

OUT&[#]n,wert[,wert2...]
OUT%[#]n,wert[,wert2...]

Gibt 16 Bit bzw. 32 Bit aus. Dieser Befehl ist erst ab Version 3.04 verfügbar.

OUT?(n)

Hierdurch wird der Ausgabestatus eines Peripheriegerätes überprüft (FALSE, wenn das Gerät nicht bereit ist).

POS(x)

Gibt die Anzahl der seit dem letzten Carriage Return auf dem Bildschirm ausgegebenen Zeichen zurück.

PRINT [werte]
PRINT AT(spalte,zeile);[werte]

Ausgabebefehl für die Ausgabe auf dem Bildschirm. Mit PRINT AT kann vor der Ausgabe der Cursor positioniert werden. Mit »werte« können Variablen, Zeichenketten und Tabulatoranweisungen angegeben werden. Ein Semikolon (;) unterbindet die Ausgabe von Carriage Return. PRINT kann auch durch ein Fragezeichen (?) abgekürzt werden.

PRINT#n,werte

Gibt über den Datenkanal »n« die angegebenen Werte aus. Der Befehl wirkt wie PRINT.

PRINT USING format$,werte
PRINT AT(zeile,spalte);USING format$,werte

Hierdurch können formatierte Ausgaben auf dem Bildschirm erfolgen. Dabei wirken die Befehle wie PRINT oder PRINT AT, jedoch wird im Formatstring das gewünschte Ausgabeformat bestimmt.

PRINT#n USING format$,werte

Dient der formatierten Ausgabe von Werten auf den Datenkanal »n«. Sonst wie PRINT USING.

PUT #n[,satz]

Ausgabe eines Datensatzes »satz« oder des nächsten Datensatzes in eine relative Datei mit der Dateinummer »n«.

READ variable[variable2...]

Mit READ werden die hinter der DATA-Anweisung stehenden Daten in Variablen übernommen. Siehe auch RESTORE.

RECALL #n,string$(),anzahl_von[TO anzahl_bis],x

Durch RECALL kann ein Stringfeld aus einer Datei eingelesen werden. Mit »anzahl_von« wird angegeben, wie viele Strings, getrennt durch Carriage Return/Line Feed, eingelesen werden sollen. In »x« steht anschließend die tatsächliche Anzahl. Es kann auch ein Bereich (anzahl_von bis anzahl_bis) von Strings angegeben werden.

RECORD #n,satz

Hierdurch wird die nächste Datensatznummer für den nächsten GET- oder PUT-Befehl eingestellt.

RELSEEK #n,anzahl

RELSEEK addiert »anzahl« zum Filepointer der Datei mit der Dateinummer »n«.

RENAME alt$ AS neu$

Dieser Befehl entspricht dem Befehl NAME.

RESTORE [marke]

Setzt den DATA-Zeiger an die Marke oder an die erste DATA-Zeile.

RMDIR name$

Löscht ein Inhaltsverzeichnis (Ordner), wenn dieses keine Dateien enthält.

SEEK #n,position

Hierdurch wird der Filepointer der Datei mit der Dateinummer »n« an »position« gesetzt.

SETMOUSE x,y[,k]

Positioniert die Maus an die angegebene Position. Dabei kann auch der Druck der Maustasten simuliert werden.

SHOWM

Schaltet die Darstellung des Mauscursors ein.

SOUND stimme,volumen,note,oktave,verzoegerung
SOUND stimme,volumen,#kombination,verzoegerung

Hierdurch wird ein Ton über die »stimme« (1 bis 3) mit der Lautstärke »volumen« (1 bis 15), der »note« (1 bis 12) und der »oktave« (1 bis 8) eingestellt. Statt »note« und »oktave« kann auch ein kombinierter Wert »kombination« (=ROUND(125000/Frequenz) angegeben werden. »verzoegerung« gibt in 50stel-Sekunden an, wann der nächste Befehl ausgeführt werden soll.

STICK 0

Schaltet die Mausabfrage am Port 0 ein.

STICK 1

Schaltet die Joystickabfrage für Port 0 und Port 1 ein.

STICK(port)

Fragt den Joystick am betreffenden Port ab.

STORE #n,x$()[,anzahl_von [TO anzahl_bis]]

Hierdurch werden »anzahl_von« oder, falls angegeben, der Bereich von »anzahl_von« bis »anzahl_bis«, Strings aus »x$« über die Datei mit der Dateinummer »n« ausgegeben.

STRIG(port)

Ermittelt den Status des Feuerknopfes eines angeschlossenen Joysticks.

TAB(anzahl)

Gibt »anzahl« Space aus.

TOUCH[#]n

TOUCH stellt Datum- und Zeiteintrag der Datei mit der Dateinummer »n« auf das aktuelle Systemdatum ein.

VTAB(zeile)

Positioniert den Cursor in die Zeile.

WAVE stimme,huell,form,dauer,verzoegerung

Durch WAVE können mehrstimmige Töne erzeugt werden.

WRITE werte

Mit WRITE werden Werte durch Kommas getrennt und Strings in Anführungszeichen ausgegeben. Dies ist normalerweise nur für Dateien sinnvoll (siehe nächsten Befehl).

WRITE #n,werte

Dient der Ausgabe von vorzugsweise mehreren Werten (numerische Werte sowie Strings) in eine Datei mit der Dateinummer »n«.

Programmsteuerung

AFTER zeit GOSUB procedure
AFTER STOP
AFTER CONT

Die erste Syntax bewirkt den Aufruf einer Prozedur nach Ablauf von »zeit« in 200stel-Sekunden. Die Prozedur wird nur einmal aufgerufen (siehe auch EVERY). Mit AFTER STOP kann diese Zeitüberwachung unterbrochen und mit AFTER CONT wieder eingeschaltet werden.

CASE y[TO z]
CASE y[,z...]
CASE TO y
CASE y TO

Verzweigungsbedingung für SELECT. Der hinter SELECT angegebene Wert x wird jeweils mit CASE überprüft. Ist die Bedingung erfüllt, wird der Programmteil hinter dem CASE-Befehl abgearbeitet, sonst folgt die nächste Überprüfung oder das Ende der Schleife. Die erste Syntax überprüft, ob x gleich y ist oder zwischen y und z liegt. Die zweite Syntax überprüft jeweils auf feste Werte. Die dritte Bedingung ist erfüllt, wenn x = y, und die vierte bei x >= y.

CHAIN name$

Durch CHAIN wird das GFA-Basic-Programm mit dem »namen$« eingeladen und gestartet.

CONT

CONT überspringt in einer SELECT-CASE-Struktur das nachfolgende CASE.

DEFFN function[(werte)]=ausdruck

Definiert eine einzeilige Funktion. Mit »werte« können Variablen (auch Strings) übergeben werden, die in der Formel »ausdruck« verwendet werden. Siehe auch FN....

DEFAULT

Teil der SELECT-CASE-Struktur. Wenn keine CASE-Bedingung erfüllt ist, wird der Programmteil zwischen DEFAULT und ENDSELECT ausgeführt. DEFAULT entspricht OTHERWISE.

DELAY zeit

Unterbricht ein Programm für »zeit« Sekunden.

DO
...
LOOP

Endlosprogrammschleife. Diese Schleife kann nur durch EXIT IF oder GOTO verlassen werden.

DO WHILE bedingung
...
LOOP oder
DO
...
LOOP WHILE bedingung

Die Programmschleife wird nur so lange durchlaufen, wie die Bedingung erfüllt ist.

DO UNTIL bedingung
...
LOOP oder
DO
...
LOOP UNTIL bedingung

Die Programmschleife wird nur so lange durchlaufen, wie die Bedingung nicht erfüllt ist. Lesen Sie hierzu bitte auch die Hinweise im Kapitel 1.

DUMP
DUMP a$[TO name$]

DUMP dient zur Fehlersuche in Programmen. Dabei werden während des Programmablaufs alle Variableninhalte ausgegeben. Bei Verwendung des Steuerwortes »a$« können auch noch zusätzliche Informationen ausgegeben, oder in die Datei »name$« geschrieben werden.

EDIT

Ruft den Editor-Bildschirm auf.

END

Programmende.

ENDDO

Entspricht der LOOP-Anweisung der DO-LOOP-Schleife.

ENDFOR

Entspricht dem NEXT bei einer FOR-NEXT-Schleife.

ENDPROC
ENDSUB

Entspricht dem RETURN als Endekennzeichen einer Prozedur.

ENDREPEAT

Entspricht dem UNTIL bei einer REPEAT-UNTIL-Schleife.

ENDSELECT

Kennzeichnet das Ende einer SELECT-CASE-Verzweigung.

ENDWHILE

Entspricht dem WEND bei einer WHILE-WEND-Schleife.

ERR

Reservierte Variable. Sie enthält die Nummer eines evtl. aufgetretenen Fehlers.

ERR$(fehlernummer)

Übergibt den Fehlertext des Fehlers »fehlernummer«.

ERROR fehlernummer

Simuliert den Fehler mit der Fehlernummer (zum Testen der eigenen Fehlerbehandlung).

EVERY zeit GOSUB procedure
EVERY STOP
EVERY CONT

Die erste Syntax bewirkt den Aufruf einer Prozedur nach Ablauf von jeweils »zeit« in 200stel-Sekunden. Die Prozedur wird immer wieder nach Ablauf dieser Zeit aufgerufen (siehe auch AFTERY). Mit EVERY STOP kann diese Zeitüberwachung unterbrochen und mit EVERY CONT wieder eingeschaltet werden.

EXIT IF bedingung

Verläßt eine Programmschleife, wenn die Bedingung erfüllt ist.

FATAL

Diese Systemvariable erhält den Wert TRUE, wenn ein aufgetretener Fehler an einer für den Interpreter unbekannten Adresse stattfand. Eine Programmfortführung durch RESUME ist dann nicht sinnvoll.

FN function[(werte)]

Führt die unter DEFFN... definierte Funktion aus und übergibt evtl. vorhandene Werte (auch Strings).

FOR schleifenvariable = anfang [DOWN]TO ende [STEP stepwert]
...
NEXT schleifenvariable

Programmschleife, die sooft wiederholt wird, bis die Schleifenvariable beginnend beim Anfang jeweils um 1 erhöht (bzw. um 1 erniedrigt bei DOWNTO) oder jeweils um den Stepwert verändert, den »ende«-Wert überschreitet.

FUNCTION function [(werte)]
...
RETURN rueckgabewert
ENDFUNC

Dies ist eine mehrzeilige Funktion (DEFFN... ist eine einzeilige Funktion). Auch hier können Übergabewerte definiert werden. Hinter RETURN wird ein »rueckgabewert« angegeben. RETURN kann auch mehrmals angegeben werden. Aufruf der Funktion erfolgt durch »FN function«, durch »@function« oder lediglich durch Angabe des Funktionsnamens.

GOSUB prozedur[(werte)]

Ruft die Prozedur mit dem Namen »procedur« auf und übergibt eventuell nötige Werte. GOSUB kann auch durch den »Klammeraffen« (@) abgekürzt werden.

GOTO marke

Springt zur Marke.

IF bedingung [THEN]
...
[ELSE][IF bedingung2]
...
ENDIF

Ist die Bedingung erfüllt, wird der Programmteil hinter THEN ausgeführt. Wenn die Bedingung nicht erfüllt ist, wird nach dem ENDIF bzw. nach ELSE, falls vorhanden, fortgefahren. Durch ELSE IF kann die IF-THEN-Schleife übersichtlicher gestaltet werden.

LIST[name$]

Listet das im Speicher befindliche Programm oder die Datei »name$«.

LLIST[name$]

Wie LIST, jedoch geht die Ausgabe zum Drucker.

LOAD name$

Lädt ein GFA-Basic-Programm (der Version 3.xx, andere Dateien müssen mit MERGE eingeladen werden). Programme, die ab Version 3.04 abgespeichert wurden, können nicht mit älteren Versionen geladen werden (siehe dazu README.304).

LOCAL variable[,variable2...]

Definiert lokale Variablen innerhalb einer Prozedur.

NEW

Löscht das im Speicher befindliche Programm.

ON BREAK
ON BREAK CONT
ON BREAK GOSUB procedure

Bestimmt die Reaktion auf das Drücken der Tasten [Control]+[SHIFT]+[Alternate]. Die erste Syntax schaltet den normalen Programmabbruch ein, während die zweite Syntax diese Unterbrechungsmöglichkeit abschaltet. Die dritte Syntax bewirkt den Aufruf der Prozedur, wenn die Tasten gedrückt werden.

ON ERROR
ON ERROR GOSUB procedure

Mit ON ERROR GOSUB... wird bei einem eventuell auftretenden Fehler die Prozedur aufgerufen. ON ERROR schaltet diese Funktion wieder ab. Siehe auch RESUME.

ON wert GOSUB procedure1,procedure2...

Je nach Größe von »wert« wird die »procedure1« (wert=1), »procedure2« (wert=2) usw. aufgerufen.

OTHERWISE

Entspricht DEFAULT.

PAUSE zeit

Bewirkt eine Programmunterbrechung von »zeit« in Fünfzigstel-Sekunden. Siehe auch DELAY.

PROCEDURE procedure[(werte)]
...
RETURN

Definiert ein Unterprogramm (Prozedur). Auch hier können Werte übergeben werden. Das Ende einer Prozedur muß immer mit RETURN gekennzeichnet werden. Die Prozedur kann mit »GOSUB procedure«, mit »@procedure« oder nur durch Angabe des Prozedurnamens aufgerufen werden.

PSAVE name$

Hierdurch wird das Programm mit einem Listschutz versehen und unter »namen$« abgespeichert. Das Programm startet dann nach dem Einladen automatisch und kann nach einem Programmabbruch nicht gelistet werden (allerdings mit SAVE wieder gespeichert und dann zum Teil gelistet werden, wobei der Rechner in der Regel abstürzt).

QUIT[n]

Durch QUIT wird das GFA-Basic verlassen. Dabei kann dem aufrufenden Programm ein Parameter übergeben werden, der angibt, ob das GFA-Basic fehlerfrei verlassen wurde (0). QUIT gleicht SYSTEM.

REM text

Durch REM kann innerhalb des Programms Kommentartext eingefügt werden. Am Zeilenbeginn kann REM auch durch ein Hochkomma ('), am Zeilenende durch ein Ausrufezeichen ersetzt werden.

REPEAT
...
UNTIL bedingung

Programmschleife, die so oft wiederholt wird, bis die Bedingung erfüllt ist.

RESUME[marke]

Bewirkt, daß nach einem Programmfehler an der Stelle des Programms fortgefahren wird, an der der Fehler auftrat (der Befehl, der den Fehler hervorrief, wird erneut ausgeführt). Es kann auch die Marke angesprungen werden.

RESUME NEXT

Durch RESUME NEXT wird an der Stelle des Programms fortgefahren, die auf den fehlerverursachenden Befehl folgt.

RUN[name$]

Starten eines Programms, das sich im Speicher befindet, oder zuvor mit »name$« eingeladen wird.

SAVE name$

Speichert ein Programm unter »name$« ab. Programme, die mit der Version ab 3.04 gespeichert wurden, können nicht mit älteren GFA-Versionen geladen werden (siehe dazu README.304).

SELECT wert

Ermöglicht eine Programmverzweigung in Abhängigkeit von »wert«. Hierzu gehören auch CASE, DEFAULT und ENDSELECT.

STEP

Siehe FOR-NEXT.

STOP

Unterbricht ein Programm.

SUB

Entspricht PROCEDURE.

SYSTEM[n]

Beendet das GFA-Basic. Siehe auch QUIT.

TRACE$

Siehe TRON procedure.

TROFF

Schaltet den Tracemodus (Ausgabe der Befehle auf dem Bildschirm während der Programmausführung) ab.

TRON
TRON #n
TRON procedure

Schaltet den Tracemodus ein. Die Ausgabe der Befehle kann auch auf die Datei mit der Dateinummer »n« erfolgen. Wird ein Prozedurnamen angegeben, wird vor jedem Befehl die Prozedur aufgerufen. Hier kann durch Auswerten der Systemvariablen TRACE$ (enthält den Befehl, der als nächster abgearbeitet werden soll) entsprechend gehandelt werden.

VAR

Mit VAR werden Variablen gekennzeichnet, die an Prozeduren oder Funktionen übergeben werden sollen.

WHILE bedingung
...
WEND

Programmschleife, die so oft durchlaufen wird, wie die Bedingung erfüllt ist.

Grafik

BITBLT source%(),destination%(),parameter%()

Ermöglicht das Kopieren von rechteckigen Bildausschnitten innerhalb beliebiger Speicherbereiche, also auch aus dem Bildschirm heraus oder in den Bildschirm hinein.

BOX x1,y1,x2,y2

Stellt ein Rechteck mit den angegebenen Koordinaten dar.

BOUNDARY wert

Schaltet die Umrandung ein (wert<>0) oder aus (wert=0).

CIRCLE x,y,r[,w1,w2]

Stellt einen Kreis an den angegebenen Koordinaten mit dem Radius »r«, oder ein Kreissegment mit Anfangswinkel »w1« und Endwinkel »w2« (in 10tel-Grad) dar.

CLIP x,y,breite,hoehe[OFFSET x0,y0]
CLIP x1,y1 TO x2,y2[OFFSET x0,y0]
CLIP #n[OFFSET x0,y0]
CLIP OFFSET x0,y0
CLIP OFF

Mit CLIP wird der Ausgabebereich (Clipping Rectangle) für die Grafikbefehle (nicht für die Line-A-Routinen) festgelegt. Dabei kann mit OFFSET auch ein neuer Koordinatenursprung definiert werden.

CLS

Löscht den Bildschirm.

COLOR farbe

Schaltet auf die Farbe des Farbregisters um. Bestimmt die Farbe für Linien und Punkte (nicht für Füllmuster oder Texte).

DEFFILL [farbe],[stil],[muster]
DEFFILL [farbe],muster$

Dieser Befehl bestimmt die Farbe sowie das Füllmuster für die Befehle PBOX, PCIRCLE, PELLIPSE, POLYFILL und FILL. Es kann außerdem ein selbstdefiniertes Muster als Bitmuster in »muster« angegeben werden.

DEFLINE [stil],[dicke],[anfang,ende]

Bestimmt die Linienart für die Befehle LINE, BOX, CIRCLE, ELLIPSE und POLYLINE.

DEFMARK [farbe],[art],[groesse]

Bestimmt das Aussehen der mit POLYMARK zu setzenden Punkte.

DEFMOUSE maus
DEFMOUSE muster$

Hierdurch wird das Aussehen des Mauscursors bestimmt. Es ist auch möglich, ein eigenes Symbol zu definieren.

DEFTEXT [farbe],[attribut],[winkel],[hoehe],[fontnummer]

Bestimmt das Aussehen eines mit TEXT auszugebenden Textes.

DRAW ausdruck

Zeichnet eine Grafik, deren Aussehen durch Befehle in »ausdruck« definiert wird.

wert=DRAW(nummer)

Liefert Informationen über den imaginären Zeichenstift, der durch den DRAW-Befehl gesteuert wird.

DRAW x1,y1[TO x2,y2][TO x3,y3]...
DRAW [TO][x,y]

Zeichnet einen Punkt oder eine oder mehrere Linien mit den angegebenen Koordinaten.

ELLIPSE x,y,rx,ry[,w1,w2]

Stellt eine Ellipse oder ein Ellipsensegment dar.

FILL x,y[,farbe]

Füllt eine abgeschlossene Fläche mit dem mit DEFFILL definierten Füllmuster. Mit »farbe« kann eine beliebige Farbe angegeben werden, die das Füllen begrenzt.

GET x1,y1,x2,y2,string$

Speichert den Bildausschnitt mit den angegebenen Koordinaten in »string$«.

GRAPHMODE mode

Bestimmt den Grafikmodus für die meisten Grafikbefehle.

LINE x1,y1,x2,y2

Zeichnet eine Linie.

PBOX x1,y1,x2,y2

Zeichnet eine gefüllte Box.

PCIRCLE x,y,r[,w1,w2]

Zeichnet einen ausgefüllten Kreis oder ein ausgefülltes Kreissegment.

PELLIPSE x,y,rx,ry[,w1,w2]

Zeichnet eine ausgefüllte Ellipse oder ein ausgefülltes Ellipsensegment.

PLOT x,y

Zeichnet einen Punkt. Entspricht DRAW x,y.

wert=POINT(x,y)

Ermittelt die Farbe eines Punktes mit der angegebenen Koordinaten.

POLYLINE anzahl,x(),y()[OFFSET xoffs,yoffs]

Stellt ein Vieleck dar, mit »anzahl« Ecken, deren Positionen in den Feldern x( ) und y( ) enthalten sind. Dabei kann ein Offset angegeben werden.

POLYFILL anzahl,x(),y()[OFFSET xoffs,yoffs]

Wie bei POLYLINE wird ein Vieleck dargestellt. Hier wird die Figur jedoch mit einem Füllmuster (mit DEFFILL definiert) ausgefüllt.

POLYMARK anzahl,x(),y()[OFFSET xoffs,yoffs]

Markiert ein Vieleck mit »anzahl« Ecken mit den Markierungssymbolen, die mit DEFMARK eingestellt wurden.

PRBOX x1,y1,x2,y1

Zeichnet ein gefülltes Rechteck mit abgerundeten Ecken.

PUT xpos,ypos,bild$[,modus]

Zeichnet einen mit GET ausgeschnittenen rechtwinkligen Bildteil an die angegebene Position. Dabei kann ein Verknüpfungsmodus (0 bis 15) angegeben werden. Lesen Sie hierzu bitte den speziellen Abschnitt im Grafikkapitel dieses Buches.

RBOX x1,y1,x2,y2

Zeichnet ein Rechteck mit abgerundeten Ecken.

SETCOLOR register,rot,gruen,blau
SETCOLOT register,mischwert

Definiert die Farbe eines Farbregisters (je nach Auflösung zwei, vier oder 16 Register).

SETDRAW xpos,ypos,winkel

Kurzform des Befehls DRAW "MA",x,y,"TT",w.

SGET bild$

Speichert den gesamten Bildschirm in die Stringvariable »bild$«.

SPRITE muster$[x,y]

Stellt ein Sprite (Spritedaten in »muster$«) dar.

SPUT bild$

Kopiert den Inhalt des Strings »bild$« auf den Bildschirm.

TEXT x,y,[,laenge],text

Stellt einen Grafiktext an der angegebenen Position dar.

VSETCOLOR register,rot,gruen,blau
VSETCOLOT register,mischwert

Definiert die Farbe eines Farbregisters (je nach Auflösung zwei, vier oder 16 Register). Dieser Befehl entspricht SETCOLOR, die Farbregisterzuordnung ist aber anders, weil hier ein VDI-Aufruf genutzt wird.

VSYNC

Auf einen senkrechten Strahlrücklauf des Monitors warten.

Line-A-Routinen

ACHAR ascii,x,y,groesse,stil,winkel

Ausgabe von einem Zeichen mit dem angegebenen ASCII-Code. »groesse« kann Werte von 0 bis 2 annehmen.

ACLIP flag,xmin,ymin,xmax,ymax

Setzt das Ausgaberechteck für die Line-A-Grafikausgaben (jedoch nicht für ALINE, BITBLT, HLINE, PSET und PTST).

ALINE x1,y1,x2,y2,farbe,muster,modus

Zeichnet eine Linie mit dem »muster« (16-Bit-Wert) im »modus« (wie GRAPHMODE) an die angegebene Position. »farbe« gibt das Farbregister an (mit SETCOLOR definiert).

APLOY adresse,anzahl,y0 TO y1,farbe,modus,m_adresse,muster_anz

Zeichnet ein Vieleck ohne Begrenzungslinie und füllt dieses mit einem Muster. In »adresse« sind die Koordinaten der einzelnen Eckpunkte (für »anzahl« Ecken) gespeichert. »farbe« und »modus« siehe ALINE. In »m_adresse« sind die Daten für das Füllmuster gespeichert.

ARECT x1,y1,x2,y2,farbe,m_adresse,muster_anz

Stellt ein Rechteck dar. Parameter siehe APLOY.

ATEXT x,y,font,text$

Gibt einen Text aus. Parameter siehe ACHAR.

BITBLT adresse%
BITBLT x%()

BITBLT-Routine der Line-A-Routinen. Die Bedeutung der umfangreichen Parameter entnehmen Sie dem speziellen Abschnitt dieses Buches.

HLINE x1,y,x2,farbe,modus,m_adresse,muster_anz

Zeichnet waagerechte Linien. Parameter siehe APLOY.

L~A

Ermittelt die Basisadresse der Line-A-Variablen.

PSET x,y,farbe

Setzt einen Punkt in der angegebenen Farbe auf dem Bildschirm.

farbe=PTST(x,y)

Ermittelt die Farbe des Punktes mit der angegebenen Position.

Ereignis-, Menü- und Fensterverwaltung

ALERT symbol,text$,default,button$,auswahl

Stellt eine Alert-Box dar mit dem Symbol (0 bis 3) und dem »text$« (maximal vier Zeilen mit jeweils 30 Zeichen). »default« gibt an, welcher Button auch über die `Return`-Taste ausgewählt werden kann. In »button$« sind die Texte der Buttons enthalten und »auswahl« liefert die Nummer des gedrückten Buttons zurück.

CLEARW[#]n

Löscht das Fenster Nummer »n«.

CLOSEW[#]n

Schließt das Fenster mit der Nummer »n« oder mit dem Index »#n«.

FILESELECT [#titel$,]pfad$,default$,name$

Stellt eine Dateiauswahlbox, die sogenannte Fileselect-Box, dar. In »pfad$« wird dazu ein Suchpfad angegeben. »default$« gibt einen voreingestellten Dateinamen an. In »name$« wird der gewählte Dateiname mit komplettem Suchpfad zurückgegeben. Ab Version 3.04 kann auch die mögliche Titelzeile der Fileselect-Box des TOS 1.4 mit »#titel$« unterstützt werden.

FULLW[#]n

Stellt das Fenster »n« auf volle Bildschirmgröße ein.

INFOW[#]n,text$

Stellt in der Informationszeile des Fensters »n« den »text$« dar.

MENU(x)

Systemvariablenfeld, das alle Daten der Ereignisse enthält.

MENU x,status

Stellt den »status« des x-ten Menüeintrages eines Pull-down-Menüs ein.

MENU menu$()

Stellt ein Pull-down-Menü dar, dessen Daten in »menu$« enthalten sind.

MENU KILL

Schaltet das Pull-down-Menü und die Kontrolle durch ON MENU GOSUB... ab.

MENU OFF

Schaltet die Überschrift eines Pull-down-Menüs wieder auf Normaldarstellung um.

ON MENU [zeit]

Überwacht das Auftreten von Ereignissen. Dabei gibt »zeit« an, wie lange (in 1000stel-Sekunden) diese Kontrolle bestehen soll.

ON MENU BUTTON klicks,maus_taste,zustand GOSUB procedure

Überwacht die Maustasten. In »klicks« wird dabei angegeben, wie viele Tastendrücke erwartet werden. »maus_taste« bestimmt, welche Taste(n) kontrolliert werden sollen. Mit »zustand« wird festgelegt, welcher Tastenzustand (gedrückt oder nicht gedrückt) das Ereignis auslöst. Es wird dann zur Prozedur gesprungen.

ON MENU GOSUB procedure

Kontrolliert Pull-down-Menüs und verzweigt in die Prozedur, wenn ein Eintrag ausgewählt wurde.

ON MENU IBOX n,x,y,breite,hoehe GOSUB procedure

Kontrolliert das Betreten eines definierten rechtwinkligen Bildschirmausschnitts »n« mit dem Mauscursor. Es wird dann zur Prozedur verzweigt.

ON MENU KEY GOSUB procedure

Kontrolliert die Tastatur und verzweigt in die Prozedur, wenn eine Taste gedrückt wird.

ON MENU MESSAGE GOSUB procedure

Kontrolliert das Auftreten einer Meldung im Message-Buffer.

ON MENU OBOX n,x,y,breite,hoehe GOSUB procedure

Kontrolliert das Verlassen eines definierten rechtwinkligen Bildschirmausschnitts »n« mit dem Mauscursor. Es wird dann zur Prozedur verzweigt.

OPENW #n,x,y,breite,hoehe,attribute
OPENW n[,x,y]

Öffnet ein Bildschirmfenster mit den Attributen an den angegebenen Koordinaten.

RC_COPY quelle,x,y,breite,hoehe TO ziel,z_x,z_y[,modus]

Kopiert einen rechtwinkligen Bildschirmausschnitt zwischen verschiedenen Bildschirmspeichern, deren Anfangsadressen in »quelle« und »ziel« angegeben werden. Die Bildschirmkoordinaten des Rechtecks stehen in den übrigen Parametern. »modus« gibt den Verknüpfungsmodus an.

RC_INTERSECT(x1,y1,breite1,hoehe1,x2,y2,breite2,hoehe2)

Diese Funktion dient zur Überprüfung, ob sich zwei Rechtecke überlappen. Die Funktion liefert TRUE zurück, wenn dies der Fall ist. Dann geben »x2«, »y2«, »breite2« und »hoehe2« die Koordinaten der Schnittfläche an.

TITLEW[#]n,text$

Schreibt in die Titelzeile des Fensters »n« den »text$«.

TOPW #n

Aktiviert das Fenster »n«.

W_HAND(#n)

Diese Funktion gibt das Handle des Fensters »n« zurück.

W_INDEX(#handle)

Diese Funktion liefert die Fensternummer des Fensters mit dem GEM-Handle »handle«.

WINDTAB
WINDTAB(i,j)

WINDTAB liefert die Anfangsadresse der Fensterparametertabelle zurück. Es kann auch WINDTAB(i,j) als Systemvariablenfeld genutzt werden, das alle Informationen über die Fenster enthält.

Systemroutinen

BIOS(n[,werte])

Ruft die BIOS-Funktion Nummer »n« auf. Je nach gewünschter Funktion müssen hierbei Parameter übergeben werden. Diese können mit der Kennung »W:« oder »L:« auch als Wort oder Langwort übergeben werden.

C:adresse([werte])

Aufruf einer C- oder Assembler-Routine ab der »adresse«. Dabei können Parameter übergeben werden, die auch als Wort oder Langwort gekennzeichnet werden können (siehe BIOS).

CALL adresse([werte])

Aufruf einer C- oder Assembler-Routine ab der »adresse«. Dabei können Parameter übergeben werden, die als Langworte interpretiert werden.

CONTRL

Liefert die Adresse des VDI-Control-Feldes zurück.

EXEC modus,name$,kommando$,envir$
EXEC (modus,name$,kommando$,envir$)

Lädt ein Programm »name$« und startet es (wenn modus=0). Nach dem Programmende wird das GFA-Basic-Programm weitergeführt. So kann also aus einem GFA-Programm heraus ein anderes Programm (z.B. eine Textverarbeitung) genutzt werden und anschließend das Basic-Programm fortgeführt werden.

GDOS?

Überprüft, ob die VDI-Erweiterung GDOS geladen wurde (Ergebnis = –1) oder nicht (0).

GEMDOS(n[,werte])

Ruft die GEMDOS-Funktion Nummer »n« auf. Dabei können Parameter übergeben werden (siehe auch BIOS).

INTIN
INTOUT

Diese Funktionen liefern die Adressen des VDI-Integer-Input-Feldes bzw. des VDI-Integer-Output-Feldes zurück.

L:

Dient zur Übergabe von Parametern als Langwort (vier Byte) an Betriebssystemfunktionen.

MONITOR[x]

Ruft ein Assembler-Programm auf. Dabei kann ein Parameter »x« übergeben werden, der in das Register D0 übernommen wird.

PTSIN
PTSOUT

Diese Funktionen liefern die Adressen des VDI-Punktkoordinaten-Input-Feldes bzw. des VDI-Punktkoordinaten-Output-Feldes zurück.

RCALL adresse,register%()

Ruft ein Assembler-Programm auf. Dabei werden im Array »register%()« die Werte angegeben, die in die Datenregister, Adreßregister und User-Stack-Pointer übernommen werden. Nach Rückkehr aus dem Assembler-Programm können die Register-Werte wieder aus dem Array ausgelesen werden.

V~H
V~H=wert

Liefert das GFA-interne VDI-Handle bzw. setzt »wert« als intern genutztes Handle ein.

V_CLRWK()

Löscht den Ausgabe-Puffer.

V_CLSWK()

Diese Funktion schließt eine mit V_OPNWK( ) geöffnete Workstation.

V_CLSVWK()

Schließt eine mit V_OPNVWK( ) geöffnete Workstation.

V_OPNWK(id)
V_OPNWK(id,1111111112)

Öffnet eine Workstation mit der Gerätekennung »id« und liefert das Handle zurück.

V_OPNVWK(id)
V_OPNVWK(id,1111111112)

Öffnet eine virtuelle Workstation mit der Gerätekennung »id« und liefert das Handle zurück.

VQT_EXTEND(text$[,x1,y1,x2,y2,x3,y3,x4,y4])

Berechnet die Größe des den »text$« umgebenden Rechtecks und übergibt die Daten in das PTSOUT-Feld oder in die angegebenen Variablen.

VQT_NAME(nummer,name$)

Ermittelt die Fontnummer und den Namen des Zeichensatzes, der unter »laufende_nummer« im Speicher vorhanden ist (durch GDOS nachgeladen).

VST_LOAD_FONT(0)

Lädt die Zeichensätze, die in der Datei ASSIGN.SYS angegeben wurden, sofern sie in den zur Verfügung stehenden Speicher passen. Diese Funktion ist nur dann aufrufbar, wenn das GDOS mitgebootet wurde. Der Parameter ist immer 0.

VST_UNLOAD_FONT(0)

Entfernt die zuvor mit VST_LOAD_FONT geladenen Zeichensätze.

V_UPDWK()

Gibt die gepufferten Grafikbefehle an das angesprochene Gerät aus.

VDIBASE

Gibt die Adresse an, ab der Parameter für die VDI-Routinen vom GEM abgelegt werden.

VDISYS[n[,c_int,c_pts[,subn]]]

Ruft die VDI-Funktion »n« auf.

W:

Dient zur Übergabe von Parametern im Wort-Format (zwei Byte) an Betriebssystem-routinen.

WORK_OUT(x)

Liefert den Wert des Index »x« (0 bis 56) zurück, die bei OPEN WORKSTATION zurückgegeben werden.

XBIOS(n[,werte])

Ruft die XBIOS-Funktion »n« auf. Dabei können Parameter übergeben werden (siehe auch BIOS).

AES-Bibliotheken

ADDRIN
ADDRIN(x)

Liefert die Adresse des AES-Adreß-Input-Feldes zurück oder greift direkt auf das entsprechende Feld »x« zu.

ADDROUT
ADDROUT(x)

Liefert die Adresse des AES-Adreß-Output-Feldes zurück oder greift direkt auf das entsprechende Feld »x« zu.

APPL_EXIT()

APPL_EXIT ist im GFA-Basic nur als Dummy-Funktion vorhanden. Sie meldet eine Applikation beim AES ab.

APPL_FIND(name$)

Liefert die Identifikationsnummer der Applikation mit Dateinamen »name$«.

APPL_INIT()

Meldet das Programm als GEM-Applikation an und liefert das Handle zurück.

APPL_READ(id,anzahl,buffer_adresse)

Liest Anzahl Bytes aus dem Ereignispuffer der Applikation mit der »id« und legt diese ab »buffer_adresse« ab.

APPL_WRITE(id,anzahl,buffer_adresse)

Schreibt Anzahl Bytes aus »buffer_adresse« in den Ereignispuffer der Applikation mit der »id«.

APPL_TPLAY(mem,num,scale)

Abspielen von zuvor aufgezeichneten Benutzeraktivitäten (Mausbewegungen usw.). Diese AES-Funktion funktioniert leider nicht.

APPL_TRECORD(mem,num)

Zeichnet Benutzeraktivitäten auf (funktioniert leider nicht).

EVNT_BUTTON(clicks,mask,state[,mx,my,button,k_state])

Überwachung des Maustasten-Ereignisses. Dabei kann die Anzahl der Tastendrücke usw. kontrolliert werden. Zurückgegeben werden können die Mausposition, die gedrückte(n) Taste(n) sowie der Zustand der Tastaturumschalt-Tasten.

EVNT_DCLICK(neu,modus)

Ermittelt die Geschwindigkeitseinstellung für den Maustasten-Doppelklick bzw. stellt diese ein.

EVNT_KEYBD()

Wartet auf ein Tastatur-Ereignis und gibt den Code der gedrückten Taste zurück.

EVNT_MESAG(adr_buffer)

Wartet auf das Eintreffen einer Meldung in den Ereignispuffer.

EVNT_MOUSE(flags,mx,my,mbreite,mhoehe,mcur_x,mcur_y,button,k_state)

Wartet, bis der Mauscursor ein Rechteck betritt oder verläßt.

EVNT_MULTI(parameter)

Wartet auf das Eintreten verschiedener Ereignisse. Die »parameter« entsprechen den Werten der anderen EVENTS (EVNT_BUTTON, EVNT_KEYBD, EVNT_MESAG, EVNT_MOUSE, EVNT_TIMER).

EVNT_TIMER(zeit)

Wartet »zeit« in 1000stel-Sekunden.

FORM_ALERT(button,string$)

Stellt eine Alert-Box dar und fragt diese ab.

FORM_BUTTON(tree,obj,clicks,new_obj)

Ermöglicht die Mauseingabe in einem Formular. Diese Funktion wird von FORM_DO genutzt.

FORM_CENTER(tree,x,y,breite,hoehe)

Ermöglicht das genaue Zentrieren eines Objekts auf dem Bildschirm.

FORM_DIAL(flag,x1,y1,b1,h1,x2,y2,b2,h2)

Je nach »flag« wird ein rechteckiger Bildausschnitt reserviert oder freigegeben oder ein schrumpfendes oder ein sich ausdehnendes Rechteck gezeichnet.

FORM_DO(tree,start_obj)

Übernimmt die Kontrolle über ein Formular (Dialogbox) oder nur bestimmter Objekte innerhalb der Dialogbox.

FORM_ERROR(err)

Zeigt eine Fehlermeldung des Fehlers »err« an.

FORM_KEYBD(baumadresse,obj,next_obj,char,new_obj,next_char)

Ermöglicht Tastatureingaben in ein Formular. Diese Funktion wird von FORM_DO genutzt.

FSEL_INPUT(pfad$,name$,[button])

Stellt eine Fileselect-Box dar.

GB

Ermittelt die Adresse des AES-Parameter-Feldes.

GCONTRL

Ermittelt die Adresse des AES-Control-Feldes.

GEMSYS n

Ruft die AES-Funktion mit der Nummer »n« auf.

GINTIN

Ermittelt die Adresse des AES-Integer-Input-Feldes.

GINTOUT

Ermittelt die Adresse des AES-Integer-Output-Feldes.

GRAF_DRAGBOX(b1,h1,x1,y1,x2,y2,b2,h2[,start_x,start_y])

Ermöglicht das Verschieben eines Rechtecks innerhalb eines größeren Rechtecks mit der Maus.

GRAF_GROWBOX(x1,y1,b1,h1,x2,y2,b2,h2)

Zeichnet ein sich ausdehnendes Rechteck.

GRAF_HANDLE(zeichen_b,zeichen_h,box_b,box_h)

Liefert die Kennung der VDI-Workstation und die Ausdehnung eines Zeichens und einer Zeichenbox des Standardzeichensatzes.

GRAF_MKSTATE(mx,my,m_state,k_state)

Liefert die aktuellen Mauskoordinaten und den Maustasten- und Tastaturstatus.

GRAF_MOUSE(m_form,muster_adresse)

Definiert das Aussehen des Mauscursors. In »m_form« steht die Nummer der Mausform und in »muster_adresse« der Zeiger auf eine selbstdefinierte Form.

GRAF_MOVEBOX(breite,hoehe,start_x,start_y,ende_x,ende_y)

Diese Funktion stellt ein bewegbares Rechteck mit der »breite« und »hoehe«.

GRAF_RUBBERBOX(x,y,breite_min,hoehe_min[,end_breite,end_hoehe])

Diese Funktion stellt ein durch die Mausbewegung in der Größe änderbares Rechteck dar (jedoch nur bei gedrückter Maustaste). Die linke obere Ecke (x und y) ist dabei feststehend. Die Funktion bricht bei nicht gedrückter Maustaste ab und liefert bei Bedarf »end_breite« und »end_hoehe« zurück.

GRAF_SHRINKBOX(x,y,breite,hoehe,x2,y2,breite2,hoehe2)

Zeichnet ein schrumpfendes Rechteck mit den Anfangskoordinaten »x2«, »y2«, »breite2« und »hoehe2« und den Endkoordinaten »x«, »y«, »breite« und »hoehe«.

GRAF_SLIDEBOX(baumadresse,rahmenobjekt,sliderobjekt,flag)

Ermöglicht das Verschieben und die Abfrage der SLIDER bei gedrückter Maustaste.

GRAF_WATCHBOX(baumadresse,objekt,in_state,out_state)

Ermöglicht den Statuswechsel eines Objekts »objekt«, wenn dieses mit dem Mauscursor berührt wird.

MENU_BAR(baumadresse,flag)

Schaltet die Menüleiste ein (flag = 1) oder aus (flag = 0).

MENU_ICHECK(baumadresse,objekt,flag)

Setzt einen Haken vor dem Objekt (flag = 1) oder löscht diesen Haken (flag = 0).

MENU_IENABLE(baumadresse,objekt,flag)

Schaltet ein Objekt (Eintrag in einem Pull-down-Menü) ein (flag = 1) oder aus (flag = 0 = Eintrag in heller Schrift).

MENU_REGISTER(id,text$)

Diese Funktion trägt den »namen$« eines Accessorys mit der »id« im ersten Pull-down-Menü ein.

MENU_TEXT(baumadresse,objekt,text$)

Durch diese Funktion kann ein Menüeintrag mit der Objektnummer »objekt« den neuen »text$« erhalten.

MENU_TNORMAL(baumadresse,titelnummer,flag)

Diese Funktion schaltet einen Menütitel mit der Objektnummer »titelnummer« auf invertierte oder normale Darstellung (flag = 0 oder flag = 1).

OB_ADR(baumadresse,objekt)

Ermittelt die Anfangsadresse des Objekts innerhalb des geladenen Resource.

OB_FLAGS(baumadresse,objekt)
OB_FLAGS(baumadresse,objekt)=wert

Ermittelt den Wert des Flagregisters zum Objekt bzw. setzt einen neuen Wert in dieses Register.

OB_X(baumadresse,objekt)
OB_Y(baumadresse,objekt)
OB_W(baumadresse,objekt)
OB_H(baumadresse,objekt)

Ermittelt die Koordinaten des Objekts im Objektbaum mit der Baumadresse. Diese Werte können auch neu eingestellt werden.

OB_HEAD(baumadresse,objekt)
OB_HEAD(baumadresse,objekt)=zeiger

Ermittelt den Zeiger auf das erste Kind-Objekt des angegebenen Objekts bzw. setzt einen neuen Zeiger.

OB_NEXT(baumadresse,objekt)
OB_NEXT(baumadresse,objekt)=zeiger

Ermittelt den Zeiger auf das nächste folgende Objekt in der gleichen Ebene bzw. setzt einen neuen Zeiger ein.

OB_SPEC(baumadresse,objekt)
OB_SPEC(baumadresse,objekt)=wert

Ermittelt den Wert der Wörter 6 und 7 des Objektparameterblocks bzw. setzt den neuen Wert ein. Hier steht je nach Objekttype in der Regel ein Zeiger auf spezielle Daten zum Objekt.

OB_STATE(baumadresse,objekt)
OB_STATE(baumadresse,objekt)=wert

Ermittelt den Status eines Objekts bzw. setzt den neuen Wert ein.

OB_TAIL(baumadresse,objekt)
OB_TAIL(baumadresse,objekt)=zeiger

Ermittelt den Zeiger auf das letzte Kind eines Objekts bzw. setzt den neuen Zeiger ein.

OB_TYPE(baumadresse,objekt)
OB_TYPE(baumadresse,objekt)=type

Ermittelt den Typ eines Objekts bzw. setzt den neuen Wert ein.

OBJC_ADD(baumadresse,eltern_objekt,kind_objekt)

Fügt in einen Objektbaum mit »baumadresse« ein neues Objekt (»kind_objekt«) in Verbindung zu einem schon vorhandenen »eltern_objekt« ein.

OBJC_CHANGE(baumadresse,objekt,0,x,y,breite,hoehe,status,re_draw)

Ändert den Status eines Objekts »objekt« innerhalb eines Objektbaums und zeichnet den Teil des Objekts, der innerhalb des Rechtecks (mit »x«, »y«, »breite« und »hoehe«) liegt, neu, wenn »re_draw = 1«.

OBJC_DELETE(baumadresse,objekt)

Löscht ein Objekt mit der Nummer »objekt« aus dem Objektbaum.

OBJC_DRAW(baumadresse,start_objekt,ebenen,x,y,breite,hoehe)

Zeichnet einen kompletten Objektbaum oder nur die Teile ab »start_objekt« bis zur »Tiefe« von ebenen Unterobjekten. Die Darstellung kann auf den rechtwinkligen Bereich (x, y, breite und hoehe) beschränkt werden.

OBJC_EDIT(baumadresse,objekt,char,old_pos,flag,new_pos)

Ermöglicht die Texteingabe und das Editieren von Texten in Objekten vom Typ G_TEXT und G_BOXTEXT.

OBJC_FIND(baumadresse,star_objekt,ebenen,x,y)

Ermittelt die Objektnummer des Objekts, das bei den Bildschirmkoordinaten x und y (z.B. Mauskoordinaten) steht. Die Suche beginnt dabei bei »start_objekt« und geht bis zu »ebenen« Unterobjekte.

OBJC_OFFSET(baumadresse,objekt,x,y)

Ermittelt die absoluten Bildschirmkoordinaten eines Objekts mit der Nummer »objekt«.

OBJC_ORDER(baumadresse,objekt,neue_position)

Ändert die Position eines Objekts innerhalb der Objektebenen.

RSRC_FREE()

Gibt den durch RSRC_LOAD genutzten Speicherplatz wieder frei. Danach ist aber trotzdem MALLOC nötig.

RSRC_GADDR(type,index,adresse)

Ermittelt die Anfangsadresse eines Objekts mit der Objektnummer »index« und dem Objekttyp »type«.

RSRC_LOAD(name$)

Diese Funktion lädt eine Resource-Datei »name$«. Zuvor wird der benötigte Speicherplatz reserviert. Es muß vorher trotzdem Speicherplatz mit MALLOC eingerichtet werden.

RSRC_OBFIX(baumadresse,objekt)

Wandelt Objektkoordinaten von Zeichen- in Pixelwerte um.

RSRC_SADDR(type,index,adresse)

Hierdurch wird innerhalb einer Resource (Objektdatei) eine Adresse gesetzt.

SCRP_READ(pfad$)

Liest aus dem SCRAP-Verzeichnis.

SCRP_WRITE(pfad$)

Schreibt in das SCRAP-Verzeichnis.

SHEL_ENVRN(adresse,such$)

Bestimmt Variablenwerte im GEM-Environment.

SHEL_FIND(pfad$)

Sucht eine Datei im aktuellen Ordner, im Wurzelverzeichnis des aktiven Laufwerks und im Wurzelverzeichnis des Laufwerks »A:« und gibt den vollständigen Pfadnamen der Datei zurück, wenn die Datei gefunden wurde.

SHEL_GET(anzahl,string$)

Liest aus dem Speicher (desktop.inf) »anzahl« Bytes und legt diese im »string$« ab.

SHEL_PUT(anzahl,string$)

Schreibt »anzahl« Bytes aus dem »string$« in den GEM-Environmentspeicher.

SHEL_READ(cmd_string$,tail_string$)

Ermittelt den Namen »tail_string$« und die Kommandozeile »cmd_string$« (von »Anwendung anmelden«) eines vom Desktop aufgerufenen Programms.

SHEL_WRITE(prg,grf,gem,cmd$,name$)

Eine andere Applikation mit Namen »name$« und der Kommandozeile »cmd$« soll gestartet werden.

WIND_CALC(type,attr,x,y,breite,hoehe,x_n,y_n,breite_n,hoehe_n)

Berechnet aus der Größe des Arbeitsbereichs die Größe des Fensters und umgekehrt. Die berechneten Koordinaten sind »x_n, y_n«, »breite_n« und »hoehe_n«.

WIND_CLOSE(handle)

Schließt das Fenster mit dem »handle«.

WIND_CREATE(attr,x,y,breite,hoehe)

Meldet ein neues Fenster an, bestimmt die maximale Größe dieses Fensters und liefert das Handle zurück.

WIND_DELETE(handle)

Löscht ein Fenster und gibt den dafür reservierten Speicherbereich zurück.

WIND_FIND(x,y)

Ermittelt aus den angegebenen Koordinaten eines Fensters dessen Handle (Kennung).

WIND_GET(handle,field,w1,w2,w3,w4)

Ermittelt diverse Informationen über ein Fenster.

WIND_OPEN(handle,x,y,breite,hoehe)

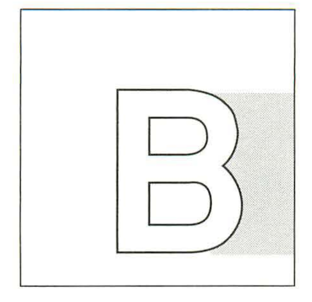
Öffnet (zeichnet) ein zuvor mit WIND_CREATE definiertes Fenster.

WIND_SET(handle,field,w1,w2,w3,w4)

Verändert Daten eines Fensters.

WIND_UPDATE(flag)

Koordiniert die Maßnahmen, die mit dem Bildschirmaufbau zusammenhängen.



Anhang

Füll-, Linienmuster, Textstil und Maussymbole

In GFA-Basic-Programmen werden häufig die sogenannten *Definitionsbefehle* für die Einstellung der Füllmuster, Linienmuster, des Text- und Mauszeigers genutzt.

Diese Befehle erwarten jeweils mehrere Parameter, die die verschiedenen Modi ein- oder ausschalten. Das Problem dabei ist, daß man sich in der Regel nicht alle einstellbaren Möglichkeiten und die dazugehörigen Parameter merken kann. Das hat zur Folge, daß immer wieder die Befehlsbeschreibung zu Rate gezogen werden muß. Dabei muß viel hin- und hergeblättert werden. Aus diesem Grunde haben wir diese wichtigen Definitionsbefehle mit allen Parametern auf den nächsten Seiten zusammengefaßt. Hier finden Sie auch das jeweilige Aussehen der gewünschten Einstellung.

DEFLINE
Syntax: DEFLINE [stil],[breite],[anfang,ende]

DEFLINE bestimmt den Linienstil, die Breite und die Anfangs- und Endform für die Befehle BOX, CIRCLE, DRAW, ELLIPSE, LINE, POLYLINE und RBOX. DEFLINE beeinflußt aber auch den Befehl PLOT.

stil = Linienstil

= 1 ————————————

= 2 — — — — — — — — — — — — — — —

= 3 . . . .

= 4 —·—·—·—·—·—·—·—·—·—·—·—·—

= 5 - - - - - - - - - - - - - - - -

= 6 -··-··-··-··-··-··-··-··-··-··-··-

= negative Zahl
(=selbstdefinierte Linie)

breite	= Linienbreite in Punkten (jedoch in Zweierschritten)
	= 1 bis 40
anfang	= Linienanfangsform
ende	= Linienendform
0	= eckig
1	= pfeilförmig
2	= rund

Beispiele:

```
DEFLINE 1,5,1,2

DEFLINE -21845,1,0,0

DEFLINE -1,1,0,0
```

DEFFILL
Syntax 1: DEFFILL [farbe],[stil],[muster]
Syntax 2: DEFFILL [farbe],muster$

Mit DEFFILL wird die Füllmusterfarbe und das Füllmuster der Befehle FILL, PBOX, PCIRCLE, PELLIPSE, POLYFILL und PRBOX bestimmt.

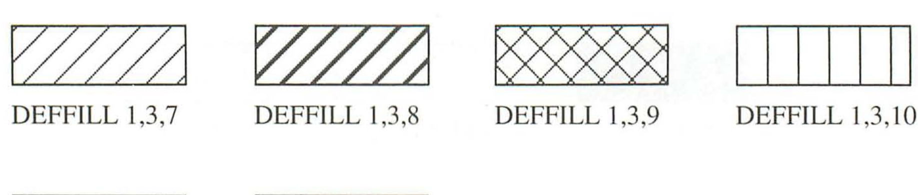
farbe	= Füllmusterfarbe (Werte von 0 bis 15)
stil	= Füllmusterstil
	= 0 (mit Hintergrundfarbe vollständig füllen)
	= 1 (mit Füllmusterfarbe vollständig füllen)
	= 2 (mit Füllmusterfarbe punktiert füllen)
	= 3 (mit Füllmusterfarbe schraffiert füllen)
	= 4 (selbstdefiniertes Füllmuster)
muster	= Füllmuster

Der Wert für »muster« ist abhängig von »stil«. Hat »stil« den Wert 2, kann »muster« die Werte 1 bis 24 erhalten. Hat dagegen »stil« den Wert 3, darf »muster« die Werte 1 bis 11 erhalten.

muster$ = Bitmuster eines selbstdefinierten Füllmusters, bestehend aus 16 Wörtern (mit MKI$ bilden), die das Bitmuster für 16 * 16 Punkte enthalten. In der mittleren Auflösung können 2 * 16 Wörter und in der niedrigen Auflösung maximal 4 * 16 Wörter angegeben werden.

Alle Füllmuster mit den dazugehörigen DEFFILL-Befehlen

DEFFILL 1,0	DEFFILL 1,1	DEFFILL 1,2,1	DEFFILL 1,2,2
DEFFILL 1,2,3	DEFFILL 1,2,4	DEFFILL 1,2,5	DEFFILL 1,2,6
DEFFILL 1,2,7	DEFFILL 1,2,8	DEFFILL 1,2,9	DEFFILL 1,2,10
DEFFILL 1,2,11	DEFFILL 1,2,12	DEFFILL 1,2,13	DEFFILL 1,2,14
DEFFILL 1,2,15	DEFFILL 1,2,16	DEFFILL 1,2,17	DEFFILL 1,2,18
DEFFILL 1,2,19	DEFFILL 1,2,20	DEFFILL 1,2,21	DEFFILL 1,2,22
DEFFILL 1,2,23	DEFFILL 1,2,24	DEFFILL 1,3,1	DEFFILL 1,3,2
DEFFILL 1,3,3	DEFFILL 1,3,4	DEFFILL 1,3,5	DEFFILL 1,3,6



DEFFILL 1,3,7 DEFFILL 1,3,8 DEFFILL 1,3,9 DEFFILL 1,3,10

DEFFILL 1,3,11 DEFFILL 1,4

DEFTEXT

Syntax: DEFTEXT [farbe],[attribut],[winkel],[hoehe],[fontnummer]

DEFTEXT bestimmt die Farbe, die Form, den Winkel und die Größe des Textes, der mit dem Befehl TEXT ausgegeben wird. Bei vorhandenem GDOS können außerdem zusätzliche Fonts genutzt werden.

farbe	= Textfarbe (Werte von 0 bis 15)
attribut	= Textform (Werte von 0 bis 31)

Text	= 0	Text	= 1
Text	= 2	Text	= 3
Text	= 4	Text	= 5
Text	= 6	Text	= 7
Text	= 8	Text	= 9
Text	= 10	Text	= 11
Text	= 12	Text	= 13
Text	= 14	Text	= 15
Text	= 16	Text	= 17
Text	= 18	Text	= 19
Text	= 20	Text	= 21
Text	= 22	Text	= 23
Text	= 24	Text	= 25
Text	= 26	Text	= 27

Text	= 28	Text	= 29
Text	= 30	Text	= 31

winkel = Darstellungswinkel (in 10tel-Grad, es sind jedoch nur vier Winkel möglich)

= 0 (normal von links nach rechts)

= 900 (von unten nach oben)

= 1800 (von rechts nach links auf dem Kopf stehend)

= 2700 (von oben nach unten)

hoehe = Darstellungsgröße (von 0 bis 26)

....	= 0	Text	= 4
Text	= 6	Text	= 10
Text	= 13	Text	= 14
Text	= 18	Text	= 22
Text	= 25	Text	= 26

DEFMOUSE

Syntax 1: DEFMOUSE symbol

Syntax 2: DEFMOUSE muster$

DEFMOUSE bestimmt das Aussehen des Mauscursors. Dazu stehen acht verschiedene vordefinierte Muster zur Verfügung.

symbol = Wert zwischen 0 und 7.

= 0

= 1

= 2

= 3

= 4

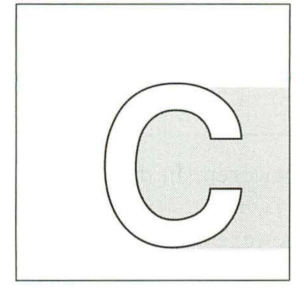
= 5

= 6

= 7

muster$ = Bitmuster für ein selbstdefiniertes Maussymbol. Die Werte müssen in Wortbreite (mit MKI$ bilden!) angegeben werden.

muster$ = MKI$ (X-Wert des Aktionspunktes)
+MKI$ (Y-Wert des Aktionspunktes)
+MKI$ (Darstellungsmodus), 1 = normal, –1 = XOR
+MKI$ (Farbe der Maske)
+MKI$ (Farbe des Vordergrunds)
+ 16 MKI$ (Bitmuster Maske)
+ 16 MKI$ (Bitmuster des Vordergrunds)



Anhang

Steuersequenzen des VT52-Emulators

Das Konsolenausgabegerät (Monitor) des Atari ST versteht einige spezielle Steuersequenzen, die sich nach der Norm der VT52-Terminals (VT52 = Video Terminal, Modell 52) richten. Die Kommandos werden mit dem Zeichen CHR$(27) eigeleitet, dem Escape-Zeichen. Man spricht daher allgemein von den Escape-Sequenzen.

Die Kommandos werden einfach mit dem PRINT-Befehl gegeben. Ein Beispiel:

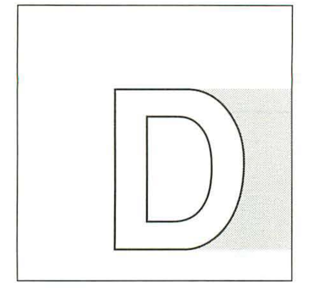
PRINT CHR$(27);"p" oder
PRINT CHR$(27);CHR$(112)

schaltet den Reverse-Modus ein (Schrift- und Hintergrundfarbe werden vertauscht). Die folgende Liste enthält alle Steuerzeichen für die Konsole.

Bezeichnung	ASCII-Wert	Beschreibung
BEL	7	Glocke. Gibt einen Ton aus.
BS	8	Backspace. Führt den Cursor eine Spalte nach links und löscht das dortige Zeichen.
HT	9	Tabulator. Setzt den Cursor auf den nächsten Tabulator-Stop (jede achte Spalte).
LF	10	Zeilenvorschub. Setzt den Cursor in die nächstfolgende Zeile.
VT	11	Vertikaler Tabulator. Setzt den Cursor ebenfalls in die nächste Zeile.
FF	12	Seitenvorschub. Setzt den Cursor ebenfalls in die nächste Zeile.
CR	13	Carriage Return. Setzt den Cursor an den Beginn der aktuellen Zeile.
ESC	27	Escape-Zeichen. Leitet eine Escape-Sequenz ein.

Bezeichnung	ASCII-Wert	Beschreibung
ESC A	27 65	Cursor eine Zeile nach oben bewegen. In der obersten Zeile ohne Wirkung.
ESC B	27 66	Cursor eine Zeile nach unten bewegen. In der untersten Zeile ohne Wirkung.
ESC C	27 67	Cursor eine Spalte nach rechts bewegen. In der letzten rechten Spalte ohne Wirkung.
ESC D	27 68	Cursor eine Spalte nach links bewegen. In der äußersten linken Spalte ohne Wirkung.
ESC E	27 69	Bildschirm löschen und Cursor in die linke obere Ecke setzen.
ESC H	27 72	Cursor in die linke obere Ecke setzen, aber den Bildschirm nicht löschen.
ESC I	27 73	Bewegt den Cursor um eine Spalte nach oben. Befindet er sich schon in der ersten Zeile, wird eine Leerzeile eingefügt und der Rest des Bildschirminhalts eine Zeile nach unten gerückt.
ESC J	27 74	Den Inhalt des Bildschirms ab der Cursorposition löschen.
ESC K	27 75	Die aktuelle Zeile ab Cursorposition löschen.
ESC L	27 76	Eine Leerzeile einfügen und den Cursor in die erste Spalte setzen. Den Rest des Bildschirms eine Zeile tiefer schieben.
ESC M	27 77	Die Zeile, in der sich der Cursor befindet, löschen und den Rest des Bildschirms eine Zeile nach oben schieben. Dadurch entsteht am unteren Rand eine Leerzeile.
ESC Y y x	27 89 y x	Positioniert den Cursor an die Zeile y und Spalte x. Die Werte werden dabei als CHR$(32+Wert) angegeben.
ESC b f	27 98 f	Schriftfarbe (0–15) festlegen. Die Farbe wird als CHR$(f) angegeben.
ESC c f	27 99 f	Hintergrundfarbe festlegen. Auch hier wird die Farbe mit CHR$(f) angegeben.
ESC d	27 100	Löscht den Bildschirm ab der linken oberen Ecke bis zur Cursorposition.
ESC e	27 101	Cursor einschalten.
ESC f	27 102	Cursor abschalten.
ESC j	27 106	Cursorposition speichern.
ESC k	27 107	Den Cursor an die mit »ESC j« gespeicherte Position setzen.

Bezeichnung	**ASCII-Wert**	**Beschreibung**
ESC l	27 108	Die aktuelle Cursorzeile löschen und den Cursor an den Zeilenanfang setzen.
ESC o	27 111	Löscht die aktuelle Cursorzeile ab dem linken Rand bis zum Cursor.
ESC p	27 112	Reverse Schrift einschalten.
ESC q	27 113	Reverse Schrift ausschalten.
ESC v	27 118	Zeilenumbruch einschalten. Der Cursor wird automatisch an den Beginn der nächsten Zeile gesetzt, wenn er über den rechten Rand hinausbewegt wird.
ESC w	27 119	Zeilenumbruch ausschalten. Es kann nicht mehr über den rechten Rand hinausgeschrieben werden.



Anhang

Anschlußbelegungen des Atari ST

Der Atari ST ist ein äußerst vielseitiger Computer. Das wird unter anderem durch die Vielzahl von Schnittstellen erreicht. Auf den folgenden Seiten zeigen wir Ihnen die Anschlußbelegungen aller Schnittstellen des Atari ST. Die Bilder zeigen die Ansicht der Anschlüsse von außen auf das Gerät.

- Maus/Joystick Port 0 (DB 9P)
- Joystick Port 1 (DB 9P)
- Stromversorgung (DIN 7P)
- MIDI Out (DIN 5S)
- MIDI In (DIN 5S)
- Monitor (DIN 13S)
- Printer/Centronics (DB 25S)
- Modem/RS232 (DB 25P)
- Floppy Disk (DIN 14S)
- DMA/Hard Disk (DB 19S)
- ROM-Modul

Maus/Joystick Port 0

Stift	Bezeichnung
1	oben (XB)
2	unten (XA)
3	links (YA)
4	rechts (YB)
5	frei
6	linke Maustaste (Feuerknopf)
7	+5 Volt
8	Masse
9	rechte Maustaste

Joystick Port 1

Stift	Bezeichnung
1	oben
2	unten
3	links
4	rechts
5	frei
6	Feuerknopf
7	+5 Volt
8	Masse
9	frei

Stromversorgung

Stift	Bezeichnung
1	+5 Volt (3 A)
2	frei
3	Masse
4	+12 Volt (30 mA)
5	–12 Volt (30 mA)
6	+5 Volt (3 A)
7	Masse

Midi-Out

Stift	Bezeichnung
1	Midi-THRU Transmit Data
2	Abschirmung
3	Midi-THRU Loop Return
4	Midi-OUT Transmit Data
5	Midi-OUT Loop Return

Midi-In

Stift	Bezeichnung
1	frei
2	frei
3	frei
4	Midi-In Receive Data
5	Midi-In Loop Return

Monitor

Stift	Bezeichnung
1	Audio-Out
2	Composite Sync
3	Frei programmierbarer Ausgang
4	Monochrom-Sensor
5	Audio-In
6	Grün
7	Rot
8	+12 Volt Schaltspannung
9	Horizontal Sync
10	Blau
11	Monochrom (Helligkeitssignal)
12	Vertical Sync
13	Masse

Printer

Stift	Bezeichnung
1	-Strobe
2	D0
3	D1
4	D2
5	D3
6	D4
7	D5

Stift	Bezeichnung
8	D6
9	D7
10	frei
11	BUSY
12–17	frei
18–25	Masse

Modem

Stift	Bezeichnung
1	Abschirmung
2	Sendedaten TxD
3	Empfangsdaten RxD
4	RTS
5	CTS
6	frei
7	Signal-Masse
8	DCD
9–19	frei
20	DTR
21	frei
22	Ring indicator (Ruf-Anzeige)
23–25	frei

Floppy Disk

Stift	Bezeichnung
1	Read Data
2	Select Seite 0
3	Masse
4	Index
5	Select Drive 0
6	Select Drive 1
7	Masse
8	Motor an

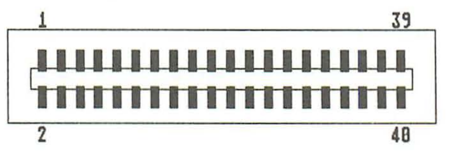
Stift	Bezeichnung
9	Direction In
10	Schritt-Impuls
11	Write Data
12	Write Gate
13	Spur 00
14	Schreibschutz

Hard Disk

Stift	Bezeichnung
1	Data 0
2	Data 1
3	Data 2
4	Data 3
5	Data 4
6	Data 5
7	Data 6
8	Data 7
9	Chip Select
10	Interrupt
11	Masse
12	Reset
13	Masse
14	ACK-Signal
15	Masse
16	A1
17	Masse
18	R/W
19	DRQ (Daten-Anforderung)

ROM-Modul

Stift	Bezeichnung	Stift	Bezeichnung
1	+5 Volt	21	A8
2	+5 Volt	22	A14
3	D14	23	A7
4	D15	24	A9
5	D12	25	A6
6	D13	26	A10
7	D10	27	A5
8	D11	28	A12
9	D8	29	A11
10	D9	30	A4
11	D6	31	ROM Select 3
12	D7	32	A3
13	D4	33	ROM Select 4
14	D5	34	A2
15	D2	35	UDS (Upper Data Strobe)
16	D3	36	A1
17	D0	37	LDS (Lower Data Strobe)
18	D1	38	Masse
19	A13	39	Masse
20	A15	40	Masse



Stichwortverzeichnis

A

Accessory-Eintrag 37
ACHAR 149
ACLIP 149
Adreßdatei 115
AES 35, 36, 54, 93
AES-Aufruf 41
AES-Bibliothek 257
AFTER CONT 46
–, STOP 46
ALERT 49
Alert-Box 164
ALINE 149
Anfangsadresse 23
APOLY 149
Array 37
ASCII-Code 51
ASCII-File 11, 169
ASCII-Wert 114
ASSIGN.SYS 237
ATEXT 149
Auflösung, hohe 24
–, mittlere 24
–, niedrige 24
Ausführungszeit 159
Auto-Ordner 218

B

Balkendiagramm 32
Basic-Compiler 160
Basic-Interpreter 160
Baumstruktur 54
Betriebssystem 22
BGET 119
Bilddatei 170
Bildschirm 23
–, logischer 239
–, physikalischer 239
Bildschirmauflösung 76
Bildschirmausgabe 134
Bildschirmausschnitt 82, 150
Bildschirmspeicher 134
BIOS 35
BITBLK-Struktur 60
BITBLT 82, 145
Bitmuster 152
BLOAD 119
BMOVE 159
BOUNDARY 145
BOX 145
BOXCHAR 57
BOXTEXT 57
BPUT 118
BSAVE 118
BUTTON 57
Buttontext 49

C

CAM 133
Chaos-Theorie 176
CHDIR 111
CHDRIVE 112
Check Mark 39
CIM 133
CIRCLE 145
CLIP 146
Clippboard 254
CLOSE 116
Closer 97
CLS 146
COLOR 139
Computer Aided Design 133
Computer Aided Manufacturing 133
Computer-Animation 133

D

Datei, relative 112
–, sequentielle 112
Dateibefehl 115
Dateiname 107
Dateiverwaltung 112
Datendatei 170
Datenkanal 117
Datenkanalnummer 120
Datensatz 114
–, Nummer 115, 131
Datenzeiger 130
DEFFILL 146
DEFILL 139
DEFLINE 146
DEFMARK 139, 146
DEFMOUSE 146
DEFTEXT 139, 146
Demoprogramm 62
Desktop 93
Desktop Publishing 133
DFREE 111
Dialogbox 53
DIR 109, 111
Diskettenbefehle 54, 107
Diskettenlaufwerk 247
DO LOOP 15
DO UNTIL 16
DO WHILE 16
DRAW 146
Drop-down-Menü 36
Drucker 108
Druckeranpassung 124
Dummy-String 37

E

Editor 10, 12
ELLIPSE 147
ELSE IF 17
Endlosschleife 32
Entscheidungsanweisungen 17
EOF 116
EVERY CONT 45
EVERY STOP 45, 53
EVNT_BUTTON 41
EVNT_KEYBD 41
EVNT_MESAG 41
EVNT_MOUSE 41
EVNT_TIMER 41
EXIST 110
EXIT IF 17

F

Farbdrucker 133
Farbgrafik 133, 137
Farbmonitor 137
FBOXTEXT 58
Fehlermeldung 170, 189
Feldelement 127
Fensterfunktion 75
Fenstergröße 99
Fensternummer 78
Fensterprogrammierung 71, 75
Fensterverwaltung 73
Festplatte 107, 159
FIELD 121
FILES 109
Filesystem, hierarchisches 107
FILL 147
Filter 144
Flag 70
Fließkommaberechnung 160
Fluchtpunkt 154
Font-Header 219
Fontnummer 237
FOR TO STEP 16
FORM_CENTER 66
FORM_DIAL 66
Fraktal 176
FTEXT 58
Füllmuster 32, 84, 317
Funktion, mehrzeilige 13
Funktionen 13

G

GDOS 217
GEM 35
GEM-Elemente 189
GEM-Zeichensatz 223
GEMDOS 25, 35
Gerätetreiber 219
GET 121
GFA-Array 42
GFA-Basic 3.03 10
GFA-Basic 3.04 10
GOSUB 13
Grafikauflösung 133
Grafikbefehle 145
Grafikbildschirm 74
Grafikdaten 60
Grafikobjekt 60, 200
Grafikspeicher 96

GRAFMODE 139
GRAPHMODE 134, 147
Grundfarben 138

H

HLINE 149
Horizontal-Offset-Tabelle 222
Hüllkurve 27
Hüllkurvenfrequenz 28

I

IBOX 57
ICON 58
ICONBLK-Struktur 58
IF THEN 17
IMAGE 57
Info-Dialogbox 181
Infozeile 86
INLINE-Befehl 32
INP 118
INPUT 118
Integerberechnung 160
Integerwert 161
Interrupt 145
Iteration 178

K

Kavalierperspektive 154
KILL 110
Komponentenmaske 78
Kontrollfeld 43

L

Label 13
Langwort 83
Laserdrucker 133
Laufwerk 107, 109
–, aktuelles 111
Lautstärke 27
LCD-Schicht 144
Leerstring 37
Lichtquelle 156
LINE 174
Line-A-Routine 145, 257
Linienmuster 317
LOCAL 13
Logbase 24
Logische Operatoren 267
Long-Integer-Feld 82
LOOP UNTIL 16
LOOP WHILE 16
Löschmodus 123

M

MALLOC 23
Mandelbrot B. 176
Mandelbrot-Menge 178
Maschinenprogramm 240
Maschinensprache 145
Matrixdrucker 133
Mauscursor 40, 44
Maustaste 70
MC 68000 145
MENU 39
MENU KILL 52
MENU OFF 52
MENU() 42
Menüeintrag 39
Menüleiste 38
Menüstruktur 49
Menüzeile 10
Message-Buffer 80, 103
MFREE 23
MIDI-Schnittstelle 108
Mikroprozessor 160
MKDIR 110
Monitor, monochromer 23
Musikprogrammierung 31
Musikuntermalung 267
Musikverarbeitung 32

N

Nachrichtennummer 92
Notentabelle 28
Nullbyte 70

O

Objektdatei 206
Objektnummer 50
Objektspeicher 207
Objekt-Statusregister 69
Objektstrukturen 54, 56, 255
Objekttyp 56
OBJ_DRAW 69
OB_FLAGS 55
OB_H 55
OB_HEAD 54

OB_NEXT 54
OB_SPEC 55
OB_STATE 55, 69
OB_TAIL 54
OB_TYPE 55
OB_W 55
OB_X 55
OB_Y 55
ODER-Verknüpfung 140
ON GOSUB 19
ON MENU BUTTON 43
ON MENU GOSUB 52
ON MENU INBOX 44
ON MENU KEY 52
ON MENU OBOX 44
ON MESSAGE GOSUB 46
OPEN 115
OPENW 38
Ordner 107
OUT 117

P

Parameterblock 56
PBOX 147
PCIRCLE 147
PEEK 159
PELLIPSE 147
Periodenwert 29
Physbase 24
Pixel 94
PLOT 147
Plotter 217
POINT 164
Pointer 116, 117
POLYFILL 148
POLYLINE 147
POLYMARK 148
Positionsflag 51
PRINT 117
PRINT-Befehl 53
Programmschleife 15
Programmsteuerung 258
Proportional-Zeichensatz 222
Prozeduren 13
–, Aufruf 14
Pull-down-Menü 197
PUT 121, 148

R

Radio-Button 69
RAM-Disk 107
Rasterzeile 219
Rauschfrequenz 27
Raytracing 156
RBOX 148
RCALL 240
RCS2.PRG 204
RC_COPY 160, 238
RECALL 120
RECORD 121
Redraw_Message 96
Reflexion 164
Register 28
Registernummer 28
Rekursion 178
RELSEEK 117
REM-Zeile 51
REPEAT UNTIL 15
RESERVE 22
Reservierte Variablen 266
Resource Construction Set 53, 251, 398
Resource-Datei 189
RETURN 14
RMDIR 111
Rückgabeparameter 43
Rückgabevariable 63

S

S/W-Monitor 141
Scan-Code 51
Schnittstelle 108
–, parallele 108
–, serielle 108
Schreibzugriff 115
Scroll-Routine 102
SEEK 117
SELECT CASE 18
SETCOLOR 138, 148
SETDRAW 148
SGET 148
Slider 74, 98
Sliderposition 99
Speicherausbau 23
Speicherbedarf 114
Speicherbereich 23, 74
Speichergröße 22
Speichermedium 121
Speicherorganisation 85
Speicherplatz 74
Speicherverschiebung 159
Speicherverwaltung 258
SPRITE 148

Sprungziel 50
SPUT 148
Stepwert 162, 195
Steuerdatei 219
STORE 119
Strahlverfolgung 156
STRING 58
Stringarray 120
Stringvariable 152
Strukturbefehl 15
Suchpfad 107
–, aktueller 107
Systemroutinen 257
Systemtimer 45
Systemvariable 164
Systemzeit 12

T

Tastaturprozessor 108
Tastaturereignis 53
Tastaturklick 32
Tedinfo-Struktur 57
TEXT 57, 148
Textverarbeitung 218
Textzeile 49
Timerzeit 41
TITEL 58
Titelzeile 86
Tondauer 33
Tongenerator 27
TOS 35
TOUCH 117
TOUCHEXIT 70
Trennzeichen 128

U

Überlappungsbereich 93
Uhrzeit 12
Unterobjekt 54
Unterprogramm 13

V

VAR-Parameter 14
Variable, lokale 13
Variablennamen 13
VDI 217
Verknüpfung 151
Video-RAM 134
VSETCOLOR 138
VSYNC 148

W

Wahrheitstabelle 139
WHILE WEND 16
WIND_CALC 77
WIND_CLOSE 79
WIND_CREATE 78
WIND_DELETE 79
WIND_GET 76
WIND_OPEN 79
WIND_UPDATE 91
WRITE 118
Wurzelverzeichnis 109

X

XBIOS 24, 35
XBIOS-Funktion 25, 134

Z

Zähler 131
Zeichen-Offset-Tabelle 219
Zeichenkettenverwaltung 274
Zeichenmatrix 219
Zeichenprogramm, objektorientiertes 199
–, pixelorientiertes 199
Zeichensatz 218
Zeiger 60
Zeilennummer 12
Zeitschriftenverwaltung 123
Zufallszahl 15
Zugriffspfad 110
2_1.GFA 23
2_2.GFA 31
3-D-Darstellung 143
3-D-Effekt 144, 156
3_1.GFA 36
3_2.GFA 62
5_1.GFA 122
6_1.GFA 141
6_2.GFA 156, 163
6_3.GFA 156, 178
6_4.GFA 204
7_1.GFA 218, 223
7_2.GFA 238
7_3.GFA 241

Software für die Atari ST





Antic Cyber Studio CAD-3D 2.0 für die Atari ST

Dieses 3-D-Zeichenprogramm ist das wichtigste Glied in der Kette der Cyber-Produkte!
Sie erstellen mit diesem Paket nicht nur dreidimensionale Objekte oder Landschaften mit verschiedenen Kontrasten und Lichtquellen, sondern auch Grafik-Animationen.
Im Programm enthalten: Cybermate. Mit Cybermate, einer leistungsfähigen und schnellen Programmiersprache, können verblüffende und raffinierte Trickfilme bearbeitet und wiedergegeben werden. Diese brillanten Berechnungsroutinen verwandeln durch trickreiche Kompressionstechniken einen 1-Mbyte-Atari in einen 10 Mbyte umfassenden Einzelbildpuffer. Sämtliche Einzelbilder eines Films von beliebig komplexen CAD-3D-Objekten werden von Cybermate im Speicher abgelegt und können zu 60 Bildern pro Sekunde abgespielt werden.

Hardware-Anforderung:
Atari-ST-Computer mit mindestens 1 Mbyte freiem RAM, Monochrom- oder Farbmonitor

3½"-Diskette
Bestell-Nr. 53102
DM 179,–* (sFr 161,–*/öS 1790,–*)

* Unverbindliche Preisempfehlung

Cyber Paint 2.0 Atari ST

Cyber Paint 2.0 ist ein Zell-Animations- und Zeichenprogramm der Spitzenklasse. Erstellen Sie 2-D-Animationen oder verarbeiten Sie dreidimensionale Cyber-Studio-Animationen weiter. Cyber Paint 2.0 arbeitet praktisch mit allen ST-Zeichenprogrammen und Aegis Animator zusammen.

Hardware-Anforderungen:
Atari ST mit 1 Mbyte RAM, Farbmonitor, 3½"-Laufwerk.
Zwei 3½"-Disketten, Bestell-Nr. 53103
DM 129,–* (sFr 116,–*/öS 1290,–*)

Zusatzprogramme zu Cyber Studio CAD-3D 2.0:

Human Design Disk, 3½"-Disk., Bestell-Nr. 53109, Future Design Disk, 3½"-Disk., Bestell-Nr. 53110, CAD-3D Plotter and Printer Drivers, 3½"-Diskette, Bestell-Nr. 53111, Architectural Design Disk, 3½"-Diskette, Bestell-Nr. 53112, CAD-3D Font Package, 3½"-Diskette, Bestell-Nr. 53114,
je DM 39,–* (sFr 35,–*/öS 390,–*)
3D Developer's Disk, 3½"-Disk., Bestell-Nr. 53113, DM 49,–* (sFr 44,–*/öS 490,–*)



Zeitschriften · Bücher
Software · Schulung

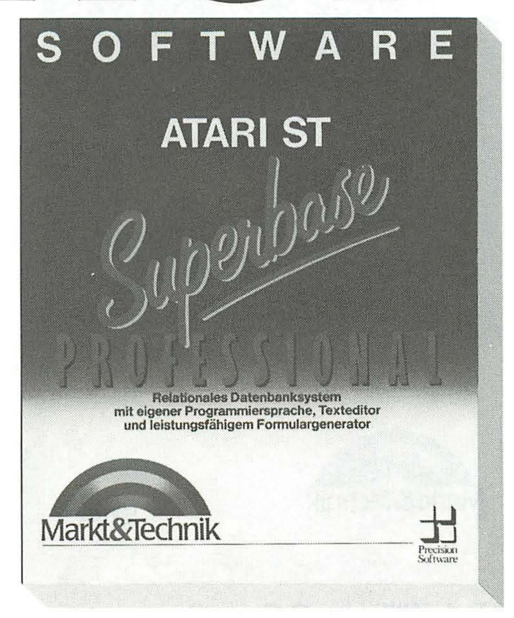
Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

511/903

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Software für die Atari ST





Superbase 2 Atari, deutsch

Superbase 2 ist die neue Version der bekannten und beliebten Datenbank Superbase. Hoher Bedienerkomfort (Pull-down-Menüs, Fenstertechnik und interaktive Eingabemasken), verbunden mit der Mächtigkeit einer relationalen Datenbank, haben Superbase zu einem der meistverkauften Systeme gemacht. Übersetzt in 9 Sprachen macht es auch in anderen Ländern die Arbeit zu einem Vergnügen.
Superbase 2 überzeugt durch zahlreiche Verbesserungen und einen neu hinzugekommenen Texteditor und Serienbrieffunktion (MailMerge), der Ihnen völlig neue Anwendungen erlaubt.
3½"-Diskette
Bestell-Nr. 53117
DM 199,–* (sFr 179,–*/öS 1990,–*)

Update Service: Superbase auf Superbase 2 (vollständig neues Handbuch)
Bestell-Nr. 53117U
DM 59,–* (sFr 59,–*/öS 590,–*)

* Unverbindliche Preisempfehlung

Superbase Professional Atari, deutsch

Superbase Professional ist die konsequente Weiterentwicklung eines der meistverkauften Datenbanksysteme der Welt. Die integrierte Programmiersprache, der Texteditor und der einzigartige Formulargenerator machen dieses Programm zum idealen Programmierwerkzeug – auch für sehr komplexe Anwendungen.
Vier 3½"-Disketten
Bestell-Nr. 51673
DM 399,–* (sFr 359,–*/öS 3990,–*)

Update Service: Superbase auf Superbase Professional Atari
Bestell-Nr. 51673U
DM 199,–* (sFr 179,–*/öS 1990,–*)

Superbase Professional Demo für Atari ST

Bestell-Nr. 53118
DM 15,–* (sFr 15,–*/öS 150,–*)

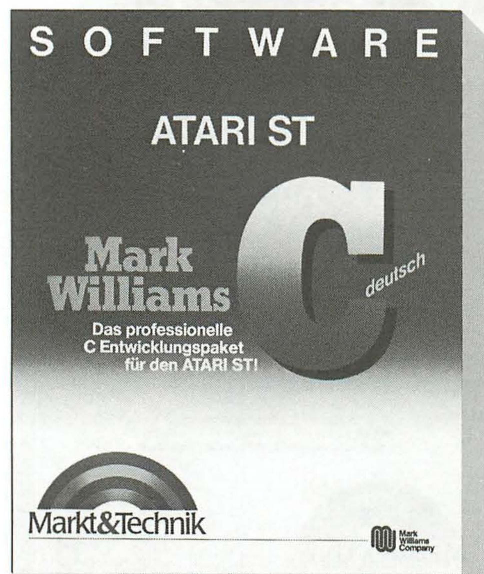


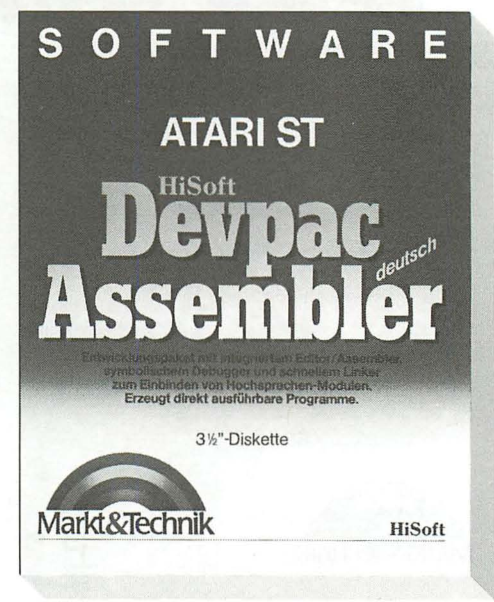
Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

512/903

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Software für die Atari ST





Mark-Williams-C-Compiler Version 3.0 (deutsch)

Ein schneller, kompakter Code mit hervorragendem Laufzeitverhalten und die hohe Zuverlässigkeit haben MW C zu dem C-Entwicklungspaket für den professionellen Programmierer gemacht. Neue Leistungsmerkmale der Version 3.0: • Anlegen von statischen Arrays, die größer als 64 Kbyte sind • Erzeugung von Objektmodulen, die mit dem Mark-Williams-C-Quelltext-Debugger csd debugged werden können • Verbesserter symbolischer Debugger • Der Resource-Editor zur Bearbeitung von Resource-Dateien • MicroEMACS Help-Funktion für die C-Programmierung • PC-relative Adressierung • Optionaler Peephole-Optimierer • Zahlreiche neue Optionen • Verbesserte Mikroshell msh.

Hardware-Anforderungen: Atari ST mit 512 Kbyte RAM, Monochrom- oder Farbmonitor, zwei 3½"-Laufwerke.
Fünf 3½"-Disketten, Bestell-Nr. 51675

DM 299,–* (sFr 269,–*/öS 2990,–*)

Update auf Mark-Williams-C 3.0:
Bestell-Nr. 51675U DM 99,–* (sFr 89,–*/öS 990,–*)

* Unverbindliche Preisempfehlung

Devpac Assembler Version 2.0

Maßgeschneidertes Entwicklungssystem für die 68000er-Programmierung. Die neuen Leistungsmerkmale der Version 2.0: • Unterstützung von lokalen Labels • Assembliert ca. 35000 Zeilen in der Minute • Makroaufruf und Inkludes können beliebig tief verschachtelt werden • Nützliche Direktiven • Volle Integration der einzelnen Teile • Assembler und Debugger können vom Editor aus per Tastendruck aufgerufen werden.

Hardware-Anforderungen: Atari ST mit ROM-TOS, Monochrom-Monitor, ein- oder zweiseitiges Laufwerk.
3½"-Diskette, Bestell-Nr. 51655

DM 148,–* (sFr 134,–*/öS 1480,–*)

Update auf Version 2.0
Bestell-Nr. 51655U, DM 19,90* (sFr 19,90*/öS 199,–*)



Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

411/812

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bitte schneiden Sie diesen Coupon aus, und schicken Sie ihn in einem Kuvert an:
Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar



Computerliteratur und Software vom Spezialisten

Vom Einsteigerbuch für den Heim- oder Personalcomputer-Neuling über professionelle Programmierhandbücher bis hin zum Elektronikbuch bieten wir Ihnen interessante und topaktuelle Titel für

• Apple-Computer • Atari-Computer • Commodore 64/128/16/116/Plus 4 • Schneider-Computer • IBM-PC, XT und Kompatible
sowie zu den Fachbereichen Programmiersprachen • Betriebssysteme (CP/M, MS-DOS, Unix, Z80) • Textverarbeitung • Datenbanksysteme • Tabellenkalkulation • Integrierte Software • Mikroprozessoren • Schulungen. Außerdem finden Sie professionelle Spitzen-Programme in unserem preiswerten Software-Angebot für Amiga, Atari ST, Commodore 128, 128 D, 64, 16, für Schneider-Computer und für IBM-PCs und Kompatible!
Fordern Sie mit dem nebenstehenden Coupon unser neuestes Gesamtverzeichnis und unsere Programmservice-Übersichten an, mit hilfreichen Utilities, professionellen Anwendungen oder packenden Computerspielen!



Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2,
8013 Haar bei München, Telefon (089) 4613-0

709005

Adresse:

Name

Straße

Ort

Bitte schicken Sie mir:
☐ Ihr neuestes Gesamtverzeichnis
☐ Eine Übersicht Ihres Programmservice-Angebotes aus der Zeitschrift

☐ Außerdem interessiere ich mich für folgende/n Computer:

(PS: Wir speichern Ihre Daten und verpflichten uns zur Einhaltung des Bundesdatenschutzgesetzes)

Markt&Technik Verlag AG
– Unternehmensbereich Buchverlag –
Hans-Pinsel-Straße 2
D-8013 Haar bei München

ATARI ST Programmierpraxis GFA-Basic 3.0

Die Autoren:

WILHELM BESENTHAL und JENS MUUS beschäftigen sich seit Jahren mit der Programmierung von Computern in Basic und Maschinensprache. Seit der Markteinführung der Atari-ST-Serie haben beide Autoren umfassende Kenntnisse über den Hardware-Aufbau und die Programmierung des Betriebssystems dieser Rechner erworben. Ihre dabei gesammelten Erfahrungen haben sie schon in mehreren erfolgreichen Büchern an den Leser weitergegeben.

Mit dem Erscheinen von GFA-Basic 2.0 im Jahre 1986 hat sich diese Programmiersprache durch einen ausgezeichneten Editor und die sehr schnelle Ausführungszeit der Programme auf dem Atari schnell als Standard etabliert. Die neue Version GFA-Basic 3.0 ist mit ihrem erweiterten, mächtigen Befehlsumfang um vieles komfortabler und leistungsfähiger geworden.

Dieses Buch enthält alle Informationen, die zum Schreiben professioneller Programme benötigt werden. Die vielseitigen Einsatzmöglichkeiten, die GFA-Basic 3.0 jedem Anwender bietet, werden dem Leser anhand zahlreicher nützlicher und interessanter Programmbeispiele gezeigt. Besonderes Augenmerk haben die Autoren dabei auf die Erstellung abgeschlossener Unterprogramme gerichtet, die der Leser sofort in seine eigenen Anwendungen einbinden kann.

Der ausführliche Anhang enthält neben der Beschreibung aller unter GFA-Basic 3.0 verfügbaren Befehle auch die Anschlußbelegungen des Atari ST sowie eine Beschreibung der Escape-Sequenzen. Damit ist dieses Buch jederzeit ein hilfreicher Begleiter für den GFA-Programmierer.

Herausragende Themen dieses Buches sind:

- Fensterprogrammierung
- Erstellen eigener Zeichensätze
- 3-D-Grafik
- Raytrace
- Zeitungsartikelverwaltung unter GEM
- Pull-down-Menüs und Dialogboxen
- Einbinden von Maschinenprogrammen
- Zeichnen auf beliebig großen Bildschirmen

Die Begleitdiskette:

- enthält alle im Buch beschriebenen Programmbeispiele

Hardware-Anforderungen:

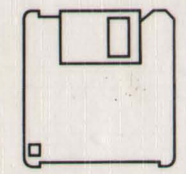
- Atari der ST-Serie mit einem Diskettenlaufwerk
- Monitor SM124

Software-Anforderungen:

- GFA-Basic-Interpreter Version 3.0

ISB N 3-89090-702-4







DM 59,-
sFr 54,30
öS 460,-